# ON THE DATA EFFICIENCY AND MODEL COMPLEXITY OF VISUAL LEARNING

by
Siyuan Qiao

A dissertation submitted to The Johns Hopkins University in conformity
with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
July 2021

# Abstract

Computer vision is a research field that aims to automate the procedure of gaining abstract understanding from digital images or videos. The recent rapid developments of deep neural networks have demonstrated human-level performance or beyond on many vision tasks that require high-level understanding, such as image recognition, object detection, *etc*. However, training deep neural networks usually requires large-scale datasets annotated by humans, and the models typically have millions of parameters and consume a lot of computation resources. The issues of data efficiency and model complexity are commonly observed in many frameworks based on deep neural networks, limiting their deployment in real-world applications.

In this dissertation, I will present our research works that address the issues of data efficiency and model complexity of deep neural networks. For the data efficiency, (i) we study the problem of few-shot image recognition, where the training datasets are limited to having only a few examples per category. (ii) We also investigate semi-supervised visual learning, which provides unlabeled samples in addition to the annotated dataset and aims to utilize them to learn better models. For the model complexity, (iii) we seek alternatives to cascading layers or blocks for improving the representation capacities of convolutional neural networks without introducing additional computations. (iv) We improve the computational resource utilization of deep neural networks by finding, reallocating, and rejuvenating underutilized neurons. (v) We present two techniques for object detection that reuse computations to reduce the architecture complexity and improve the detection performance. (vi) Finally, we show our work on reusing visual features for multi-task learning to improve computation efficiency and share training information between different tasks.

## Thesis Readers

Dr. Alan L. Yuille (Primary Advisor)
Bloomberg Distinguished Professor
Department of Computer Science
Johns Hopkins University

Dr. Wei Shen
Associate Professor
Artificial Intelligence Institute
Shanghai Jiao Tong University

Dr. Rama Chellappa
Bloomberg Distinguished Professor
Department of Electrical and Computer Engineering
Johns Hopkins University

*To my family and fiancée Wanyu Huang for their unending support.*

# Acknowledgements

First and foremost, I thank my Ph.D. advisor Prof. Alan L. Yuille. My doctoral journey started with his talks at YITU when I was an undergraduate intern there. The talks enlightened the path towards the boundary of human knowledge and presented us the value to human beings of a researcher pushing that boundary. I was so inspired and got incredibly lucky later to join his team at Johns Hopkins University. Since then, he has always been a truly kind and supportive advisor. I would like to say thank you to him for introducing me to the beautiful territory of vision research await for us to explore; for supporting me every moment when I was confused and lost; for telling me what it takes to become a responsible researcher; and for educating me to be open, kind, and love to share. The time to pursue a doctoral degree was long, but he has made it enjoyable; the time was also short, and he has taken every chance to get his students prepared for their careers. I appreciate his classes I took, every discussion and meeting we had, and memorable holiday parties with his family.

I would also like to thank my thesis committee: Alan Yuille, Wei Shen, and Rama Chellappa for their support and guidance on this dissertation. I would like to express the most sincere thanks to Wei Shen. He is a very strong and talented researcher. He was a collaborator in many of my papers during my doctoral study and has taught me many important research skills including idea brainstorming, writing academic papers, and giving conference presentations. I thank Rama Chellappa for teaching me hierarchical thinking to solve complex problems, and extending research experience and skills to real-life problems. I am also very grateful to Alan Yuille, Wei Shen, Gregory Hager, Yinzhi Cao, Vishal Patel, Rama Chellappa, and Michael Bonner for serving on my GBO committee.

I am thankful to my mentors and collaborators during internships. They are Haichao Zhang and Wei Xu at Baidu IDL; Zhe Lin, Jianming Zhang, and Yilin Wang at Adobe; Liang-Chieh Chen, Yukun Zhu, and Hartwig Adam at Google. I was extremely fortunate to have worked with them and learned from them the professional ways to balance academic and industrial impacts, think out of the box, plan things ahead and execute them in time.

I thank all members of CCVL for the research discussions as well as the parties we had together. They are Xiaodi Hou, Yukun Zhu, Yuan Gao, Liang-Chieh Chen, Xianjie Chen, Fangting Xia, Jun Zhu, Junhua Mao, Peng Wang, Jianyu Wang, Xiaochen Lian, Xiao Chu, Ehsan Jahangiri, Feng Wang, Chang Liu, Vittal Premachandran, Xuan Dong, Song Bai, Xiang Xiang, Peng Tang, Lingxi Xie, Huangjie Zheng, Zhe Ren, Chenxi Liu, Zhishuai Zhang, Yan Wang, Wei Shen, Yuyin Zhou, Weichao Qiu, Cihang Xie, Minghui Liao, Yuhui Xu, Zefan Li, Zhuotun Zhu, Adam Kortylewski, Yongyi Lu, Zongwei Zhou, Chenxu Luo, Huiyu Wang, Qing Liu, Qi Chen, Fengze Liu, Yi Zhang, Hongru Zhu, Yingda Xia, Jieru Mei, Qihang Yu, Yingwei Li, Yixiao Zhang, Zhuowan Li, Zihao Xiao, Chenglin Yang, Yutong Bai, Angtian Wang, Chen Wei, Jieneng Chen, Ju He, and Prakhar Kaushik. Special thanks to Wei Shen, Lingxi Xie, Weichao Qiu, and Chenxi Liu for being the model senior researchers when I was a blank page, and to Liang-Chieh Chen and Yukun Zhu for their support and guidance in my senior doctoral study and research. I am extremely grateful to be a member of this big family.

Thank you to my two cats, Harvey and DiDi, if they could understand human language. They encouraged me every moment in this memorable journey and certainly deserve a big pack of cat treats. Most importantly, I would like to thank my family and fiancée Wanyu Huang for always being there when I need them. I'm so grateful for their support. Without them, this fruitful journey would not be possible.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Computer vision is about solving problems to help computers see [78]. At the core of computer vision, the objective is to infer something about the world from images or videos. This something could be what is present in the image, where it is, how it is interacting with other objects, *etc*. Many computer vision problems are deceptively simple mostly because humans are so good at them that they easily overlook the complexity behind the scene. Yet only until recently have many computer vision models caught up or surpassed human-level performance on several vision tasks with abstract understandings, such as image recognition [108], object detection [107], *etc*. And one of the driving forces of the success is deep neural networks [142].

Deep neural networks achieve state-of-art performances in many vision tasks [41], [139], [214]. However, training neural networks usually requires large-scale labeled datasets which are difficult to collect [173], [230], yields models with millions of parameters which limit their deployment in real-world applications [116], and consumes a lot of computational resources such as computation time and memory [96]. These issues can be summarized as two problems: the data efficiency and the model complexity of deep neural networks when they are applied to computer vision tasks. It is not surprising to see the success of deep neural networks made possible only in the last decade given the recent vast efforts put into annotating large-scale datasets and superior advances of special computing devices. Nevertheless, the data efficiency and model complexity issues are still commonly observed in real-world vision problems, hence motivating the research works that compose this dissertation.

**Figure 1-1.** Annotation examples of the COCO dataset [173] for instance segmentation.

## 1.1 On the Data Efficiency

In the first part of the dissertation, I will discuss the data efficiency of training deep neural networks for vision tasks. Neural networks can be used to perform numerous vision tasks, including image classification [142], object detection [72], instance segmentation [173], semantic segmentation [53], *etc*. Figure 1-1 shows annotation examples of the COCO dataset [173] for instance segmentation, where pixels are labeled such that each instance is associated with a semantic class, *e.g.* a cat or a dog, and all its pixels have a unique instance ID to differentiate itself with the other instances that might have the same semantic class. Training neural networks to perform vision tasks usually requires a huge amount of annotated examples. For instance, ImageNet [142] dataset collects 14,197,122 images and 21841 synsets, COCO [173] labels

123,287 images and 886,284 instances. From the examples in Figure 1-1, we can also see that labeling a dataset is also resource-consuming: data annotators need to spend months sometimes or more to provide detailed and accurate labels. Experts are also needed sometimes for vision tasks that require domain knowledge, *e.g.* medical vision intelligence [322], which further increases the difficulty of labeling datasets for training neural networks.

The need for large datasets for training deep neural networks is rooted in how networks are trained. The most popular and successful training methods for deep neural networks are based on gradients [134], [244]. The optimization process is iterative, taking each training sample as input and use back-propagation to compute the gradients to direct the steps to minimize the loss function. Deep neural networks usually have millions of parameters to model the complexity of the vision tasks, and training them from scratch using gradient-based optimizers will need the dataset to have similar complexity. Fine-tuning is a popular technique to reduce the dataset size [214], yet the size issue still remains for the network pre-training, hence limiting the application of neural networks in many real-world problems.

I will present two of our research works on the data efficiency of training neural networks for computer vision. The first work is on the problem of few-shot learning [216]. As discussed above, the success of deep neural networks on many vision tasks relies on large-scale datasets. By contrast, humans are able to learn new concepts well from a few examples after they have accumulated enough past knowledge [22]. Few-shot learning is a topic that aims at narrowing this gap between machine and human intelligence. As its name suggests, few-shot learning studies the problem of learning from a limited number of examples. Few-shot learning methods can significantly improve the data efficiency and reduce the number of samples needed for training neural networks. It also addresses the issues in data collection when supervised examples are hard or impossible to collect due to safety, privacy, *etc*. Our work [216] proposes a novel method that can adapt a pre-trained neural network to novel categories by directly predicting the parameters from the activations, which achieved state-of-the-art performances on many few-shot learning benchmarks.

The other work is on semi-supervised learning [218]. Semi-supervised learning is about utilizing unlabeled samples in addition to labeled data to learn a better model [23], [328]. This setting is more practically appealing than the few-shot learning scenario because in many real-world cases it is hard to label the samples but the quantity of the unlabeled samples is ample. Moreover, providing additional unlabeled samples does not hurt because the fully-supervised methods can simply ignore their existence. In this dissertation, I'll present our research work Deep Co-Training [218], which brings the award-winning Co-Training [23] to deep learning and achieved state-of-the-art results on semi-supervised benchmarks.

## 1.2 On the Model Complexity

The complexity of deep neural networks has increased dramatically over the past years to suffice the need for modeling computer vision tasks [250]. The model complexity comes in two ways: the network architectures are becoming more and more complex [175], and the best-performing models are also very large in terms of the computational resource (*e.g.*, the number of parameters and FLOPs) for the current generation of computing devices [28]. In this dissertation, I will cover four of our research works on the model complexity of deep neural networks for computer vision tasks. They aim to squeeze more performance out of deep neural networks from the perspective of both architecture complexity and computational resource.

The first work focuses on improving the representation capacities of convolutional neural networks without introducing additional computational costs [219]. It is motivated by the observation that many state-of-the-art network architectures usually cascade convolutional layers or building blocks to increase the depth [108], [120], which is one of the keys to the capacity of neural networks. Our paper proposes an alternative method for increasing the depth of neural networks [219]. It introduces computation orderings to the channels within convolutional layers or blocks and computes the outputs gradually in a channel-wise manner based on the added computation orderings. The effects of the computation orderings are

two-fold: it not only increases the depths but also removes the overlap singularities within the neural networks. As a result, it increases the convergence speed and improves the performance of the neural networks. Experimental results show that the networks based on our method achieved state-of-the-art accuracies on popular image classification benchmarks.

Next, this dissertation presents our work on improving deep network training by enhancing computational resource utilization [215]. This work is motivated by the observation that deep neural networks are usually over-parameterized for their tasks, hence resulting in underutilized computational resources. Network pruning is motivated by this observation, which removes the unused channels or weights to speed up the inference [181]. Our method Neural Rejuvenation [215] approaches the problem in an opposite way: it studies the problem of improving the resource utilization of neural networks to realize the potentials of the networks. Neural Rejuvenation is designed to detect dead neurons and compute resource utilization in real time, and rejuvenate the dead neurons if the utilization is below a threshold. It is a plug-and-play module compatible with many optimizers. Neural networks trained with Neural Rejuvenation achieve better performances while maintaining the resource usages.

From the perspective of architecture complexity, this dissertation presents our work DetectoRS [214]. It proposes two techniques for object detection by reusing computations to improve detection performance. Reusing computations is also studied as the mechanism of looking and thinking twice, which is aligned well with the human visual perception that selectively enhances and suppresses neuron activations by passing high-level semantic information through feedback connections [61]. DetectoRS [214] is composed of two techniques. At the macro level, it proposes Recursive Feature Pyramid, which reuses Feature Pyramid Networks and the bottom-up backbone layers to enhance the capacity. At the micro level, it proposes Switchable Atrous Convolution, which reuses atrous convolutions at different rates to adapt to different object scales. Combining them results in DetectoRS, which achieved state-of-the-art performance on COCO test-dev for the tasks of object detection, instance segmentation, and panoptic segmentation.

Finally, this dissertation presents our work ViP-DeepLab [220] on reusing visual features for multi-task learning [138]. The advantages of sharing visual features for multi-task learning are multifold: it reduces the need for designing task-specific architectures thus decreases the architecture complexity; it speeds up the inference speed as the visual features are shared; it can lead to better performance if the tasks provide complementary and helpful training information to each other, *etc*. Our work ViP-DeepLab [220] is a unified model attempting to tackle the long-standing and challenging inverse projection problem in vision, which we model as the joint task of monocular depth estimation and video panoptic segmentation. ViP-DeepLab achieved state-of-the-art results on each individual task.

## 1.3 Outline

This dissertation is organized as follows. In Chapter 1, I introduce the topic of this dissertation, list the sub-problems, and discuss how we approach them. The remainder of the dissertation is divided into two parts: Chapter 2 and 3 focus on the data efficiency of visual learning, while Chapter 4, 5, 6, and 7 introduce methods we propose for the model complexity of visual learning. Specifically, in Chapter 2, we propose a novel method for few-shot learning where the number of categories is large and the number of examples per novel category is very limited. In Chapter 3, we study how to extend the concept of Co-Training to deep learning for semi-supervised image recognition where the dataset contains both labeled and unlabeled samples. In Chapter 4, we present a method to increase the depth of neural networks, which is an alternative to cascading convolutional layers or building blocks. In Chapter 5, we study the problem of improving computational resource utilization of neural networks to address the issue of over-parameterization. In Chapter 6, we explore the mechanism of looking and thinking twice by reusing architectures and computations in the backbone design for object detection. In Chapter 7, we present a unified model attempting to tackle the long-standing and challenging inverse projection problem in vision by reusing the visual features for multiple vision tasks. Finally, Chapter 8 concludes this dissertation.

## 1.4 Relevant Publications

The ideas in this dissertation are composed of the following publications.

1. **Siyuan Qiao**, Chenxi Liu, Wei Shen, and Alan L. Yuille. "Few-Shot Image Recognition by Predicting Parameters from Activations." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

2. **Siyuan Qiao**, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan Yuille. "Deep Co-Training for Semi-Supervised Image Recognition." In Proceedings of the European Conference on Computer Vision. 2018.

3. **Siyuan Qiao**, Zhishuai Zhang, Wei Shen, Bo Wang, and Alan Yuille. "Gradually Updated Neural Networks for Large-Scale Image Recognition." In International Conference on Machine Learning. PMLR, 2018.

4. **Siyuan Qiao**, Zhe Lin, Jianming Zhang, and Alan L. Yuille. "Neural Rejuvenation: Improving Deep Network Training by Enhancing Computational Resource Utilization." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

5. **Siyuan Qiao**, Liang-Chieh Chen, and Alan Yuille. "DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

6. **Siyuan Qiao**, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. "ViP-DeepLab: Learning Visual Perception with Depth-aware Video Panoptic Segmentation." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

The other publications during my doctoral study are listed below.

1. **Siyuan Qiao**, Wei Shen, Weichao Qiu, Chenxi Liu, and Alan Yuille. "ScaleNet: Guiding Object Proposal Generation in Supermarkets and Beyond." In Proceedings of the IEEE International Conference on Computer Vision. 2017.

2. Yan Wang, Lingxi Xie, Chenxi Liu, **Siyuan Qiao**, Ya Zhang, Wenjun Zhang, Qi Tian, and Alan Yuille. "Sort: Second-Order Response Transform for Visual Recognition." In Proceedings of the IEEE International Conference on Computer Vision. 2017.

3. Weichao Qiu, Fangwei Zhong, Yi Zhang, **Siyuan Qiao**, Zihao Xiao, Tae Soo Kim, and Yizhou Wang. "UnrealCV: Virtual Worlds for Computer Vision." In Proceedings of the 25th ACM International Conference on Multimedia. 2017.

4. Zhishuai Zhang, **Siyuan Qiao**, Cihang Xie, Wei Shen, Bo Wang, and Alan L. Yuille. "Single-Shot Object Detection with Enriched Semantics." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

5. Yan Wang, Lingxi Xie, **Siyuan Qiao**, Ya Zhang, Wenjun Zhang, and Alan L. Yuille. "Multi-Scale Spatially-Asymmetric Recalibration for Image Classification." In Proceedings of the European Conference on Computer Vision. 2018.

6. Chenglin Yang, Lingxi Xie, **Siyuan Qiao**, and Alan L. Yuille. "Training Deep Neural Networks in Generations: A More Tolerant Teacher Educates Better Students." In Proceedings of the AAAI Conference on Artificial Intelligence. 2019.

7. Zhishuai Zhang, Wei Shen, **Siyuan Qiao**, Yan Wang, Bo Wang, and Alan Yuille. "Robust face detection via learning small faces on hard images." In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2020.

8. Hao Ding, **Siyuan Qiao**, Alan Yuille, and Wei Shen. Deeply Shape-guided Cascade for Instance Segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

# Chapter 2

# Few-Shot Image Recognition by Predicting Parameters from Activations

In this chapter, we are interested in the few-shot learning problem. In particular, we focus on a challenging scenario where the number of categories is large and the number of examples per novel category is very limited, *e.g.* 1, 2, or 3. Motivated by the close relationship between the parameters and the activations in a neural network associated with the same category, we propose a novel method that can adapt a pre-trained neural network to novel categories by directly predicting the parameters from the activations. Zero training is required in adaptation to novel categories, and fast inference is realized by a single forward pass. We evaluate our method by doing few-shot image recognition on the ImageNet dataset, which achieves state-of-the-art classification accuracy on novel categories by a significant margin while keeping comparable performance on the large-scale categories. We also test our method on the MiniImageNet dataset and it strongly outperforms the previous state-of-the-art methods.

## 2.1 Introduction

Recent years have witnessed rapid advances in deep learning [154], with a particular example being visual recognition [108], [142], [239] on large-scale image datasets, *e.g.*, ImageNet [230]. Despite their great performances on benchmark datasets, the machines exhibit a clear difference with people in the way they learn concepts. Deep learning methods typically require huge

amounts of supervised training data per concept, and the learning process could take days using specialized hardware, *i.e.* GPUs. By contrast, children are known to be able to learn novel visual concepts almost effortlessly with a few examples after they have accumulated enough past knowledge [22]. This phenomenon motivates computer vision research on the problem of few-shot learning, *i.e.*, the task to learn novel concepts from only a few examples for each category [74], [147].

Formally, in the few-shot learning problem [137], [189], [260], we are provided with a large-scale set $\mathcal{D}_{\text{large}}$ with categories $\mathcal{C}_{\text{large}}$ and a few-shot set $\mathcal{D}_{\text{few}}$ with categories $\mathcal{C}_{\text{few}}$ that do not overlap with $\mathcal{C}_{\text{large}}$. $\mathcal{D}_{\text{large}}$ has sufficient training samples for each category whereas $\mathcal{D}_{\text{few}}$ has only a few examples ($< 6$ in this chapter). The goal is to achieve good classification performances, either on $\mathcal{D}_{\text{few}}$ or on both $\mathcal{D}_{\text{few}}$ and $\mathcal{D}_{\text{large}}$. We argue that a good classifier should have the following properties: (1) It achieves reasonable performance on $\mathcal{C}_{\text{few}}$. (2) Adapting to $\mathcal{C}_{\text{few}}$ does not degrade the performance on $\mathcal{C}_{\text{large}}$ significantly (if any). (3) It is fast in inference and adapts to few-shot categories with little or zero training, *i.e.*, an efficient lifelong learning system [50], [51].

Both parametric and non-parametric methods have been proposed for the few-shot learning problem. However, due to the limited number of samples in $\mathcal{D}_{\text{few}}$ and the imbalance between $\mathcal{D}_{\text{large}}$ and $\mathcal{D}_{\text{few}}$, parametric models usually fail to learn well from the training samples [189]. On the other hand, many non-parametric approaches such as nearest neighbors can adapt to the novel concepts easily without severely forgetting the original classes. But this requires careful designs of the distance metrics [9], which can be difficult and sometimes empirical. To remedy this, some previous work instead adapts feature representation to the metrics by using siamese networks [137], [174]. As we will show later through experiments, these methods do not fully satisfy the properties mentioned above.

In this chapter, we present an approach that meets the desired properties well. Our method starts with a pre-trained deep neural network on $\mathcal{D}_{\text{large}}$. The final classification layers (the fully connected layer and the softmax layer) are shown in Figure 2-1. We use $\mathbf{w}_y \in \mathbb{R}^n$ to

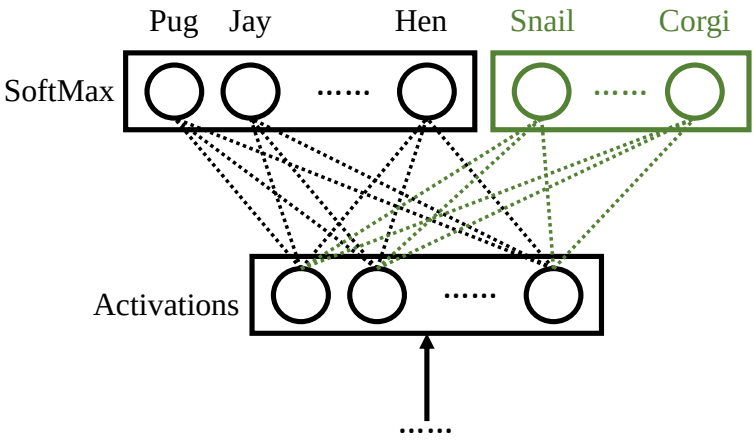



**Figure 2-1.** Illustration of pre-training on $\mathcal{D}_{\text{large}}$ (black) and few-shot novel category adaptation to $\mathcal{D}_{\text{few}}$ (green). The green circles are the novel categories, and the green lines represent the unknown parameters for categories in $C_{\text{few}}$.

denote the parameters for category $y$ in the fully connected layer, and use $\mathbf{a}(x) \in \mathbb{R}^n$ to denote the activations before the fully connected layer of an image $x$. Training on $\mathcal{D}_{\text{large}}$ is standard; the real challenge is how to re-parameterize the last fully connected layer to include the novel categories under the few-shot constraints, *i.e.*, for each category in $\mathcal{C}_{\text{few}}$ we have only a few examples. Our proposed method addresses this challenge by directly predicting the parameters $\mathbf{w}_y$ (in the fully connected layer) using the activations belonging to that category, *i.e.* $\mathcal{A}_y = \{\mathbf{a}(x) | x \in \mathcal{D}_{\text{large}} \cup \mathcal{D}_{\text{few}}, Y(x) = y\}$, where $Y(\cdot)$ denotes the category of the image.

This parameter predictor stems from the tight relationship between the parameters and activations. Intuitively in the last fully connected layer, we want $\mathbf{w}_y \cdot \mathbf{a}_y$ to be large, for all $\mathbf{a}_y \in \mathcal{A}_y$. Let $\bar{\mathbf{a}}_y \in \mathbb{R}^n$ be the mean of the activations in $\mathcal{A}_y$. Since it is known that the activations of images in the same category are spatially clustered together [64], a reasonable choice of $\mathbf{w}_y$ is to align with $\bar{\mathbf{a}}_y$ in order to maximize the inner product, and this argument holds true for all $y$. To verify this intuition, we use t-SNE [187] to visualize the neighbor embeddings of the activation statistic $\bar{\mathbf{a}}_y$ and the parameters $\mathbf{w}_y$ for each category of a pre-trained deep neural network, as shown in Figure 2-2. Comparing them and we observe a high similarity in both the local and the global structures. More importantly, the semantic structures [122] are also preserved in both activations and parameters, indicating promising generalizability to unseen

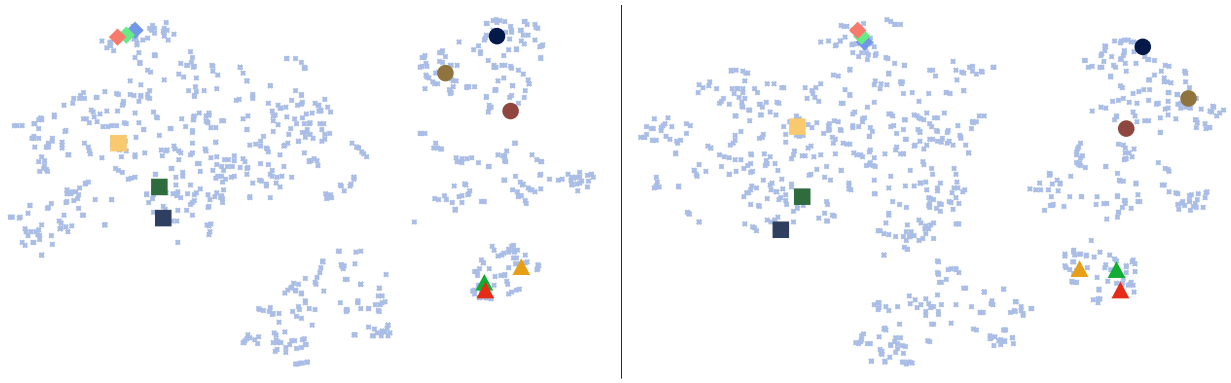


**Figure 2-2.** Our motivation: t-SNE [187] results on the average activations $\bar{\mathbf{a}}_y$ of each category before the fully connected layer of a $50$-layer ResNet [108] pre-trained on $\mathcal{D}_{\text{large}}$ from ImageNet [230] (left) and the parameters $\mathbf{w}_y$ of each category in the last fully connected layer (right). Each point represents a category. Highlighted points with the same color and shape correspond to the same category. Circles are *mammals*, triangles are *birds*, diamonds are *buses*, and squares are *home appliances.*

categories.

These results suggest the existence of a category-agnostic mapping from the activations to the parameters given a good feature extractor $\mathbf{a}(\cdot)$. In our work, we parameterize this mapping with a feedforward network that is learned by back-propagation. This mapping, once learned, is used to predict parameters for both $\mathcal{C}_{\text{few}}$ and $\mathcal{C}_{\text{large}}$.

We evaluate our method on two datasets. The first one is MiniImageNet [260], a simplified subset of ImageNet ILSVRC 2015 [230], in which $C_{\text{large}}$ has $80$ categories and $C_{\text{few}}$ has $20$ categories. Each category has $600$ images of size $84 \times 84$. This small dataset is the benchmark for natural images that the previous few-shot learning methods are evaluated on. However, this benchmark only reports the performances on $D_{\text{few}}$, and the accuracy is evaluated under $5$-way test, *i.e.*, to predict the correct category from only $5$ category candidates. In this chapter, we will take a step forward by evaluating our method on the full ILSVRC 2015 [230], which has $1000$ categories. We split the categories into two sets where $\mathcal{C}_{\text{large}}$ has $900$ and $\mathcal{C}_{\text{few}}$ has the rest $100$. The methods will be evaluated under $1000$-way test on both $D_{\text{large}}$ and $D_{\text{few}}$. This is a setting that is considerably larger than what has been experimented with in the few-shot learning before. We compare our method with the previous work and show state-of-the-art

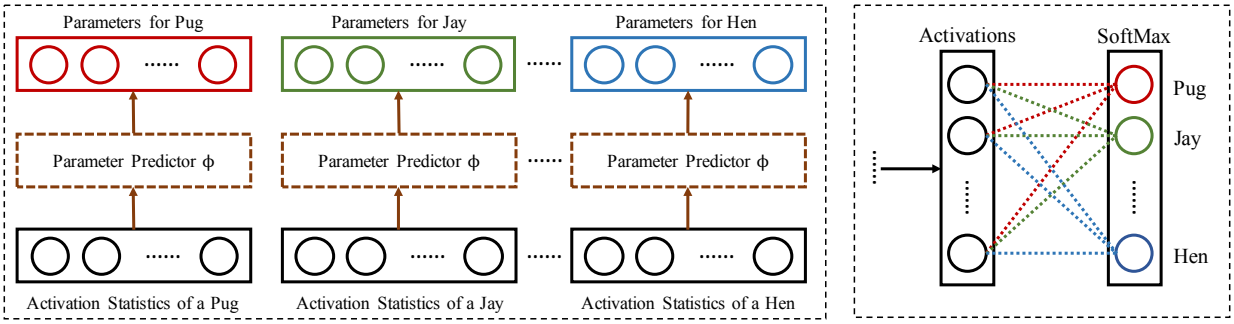



**Figure 2-3.** Building the fully connected layer by parameter prediction from activation statistics.

performances.

The rest of the chapter is organized as follows: §2.2 defines and explains our model, §2.3 presents the related work, §2.4 shows the experimental results, and §2.5 concludes the chapter.

## 2.2 Model

The key component of our approach is the category-agnostic parameter predictor $\phi : \bar{\mathbf{a}}_y \rightarrow \mathbf{w}_y$ (Figure 2-3). More generally, we could allow the input to $\phi$ to be a statistic representing the activations of category $y$. Note that we use the same mapping function for all categories $y \in \mathcal{C}_{\text{large}}$, because we believe the activations and the parameters have similar local and global structure in their respective space. Once this mapping has been learned on $\mathcal{D}_{\text{large}}$, because of this structure-preserving property, we expect it to generalize to categories in $\mathcal{C}_{\text{few}}$.

### 2.2.1 Learning Parameter Predictor

Since our final goal is to do classification, we learn $\phi$ from the classification supervision. Specifically, we can learn $\phi$ from $\mathcal{D}_{\text{large}}$ by minimizing the classification loss (with a regularizer $||\phi||$) defined by

$$\mathcal{L}(\phi) = \sum_{(y,x)\in\mathcal{D}_{\text{large}}} \left[ -\phi\left(\bar{\mathbf{a}}_y\right)\mathbf{a}(x) + \log \sum_{y'\in\mathcal{C}_{\text{large}}} e^{\phi\left(\bar{\mathbf{a}}_{y'}\right)\mathbf{a}(x)} \right] + \lambda||\phi|| \tag{2.1}$$

Eq. 2.1 models the parameter prediction for categories $y \in \mathcal{C}_{\text{large}}$. However, for the few-shot set $\mathcal{C}_{\text{few}}$, each category only has a few activations, whose mean value is the activation itself when each category has only one sample. To model this few-shot setting in the large-scale training on $\mathcal{D}_{\text{large}}$, we allow both the individual activations and the mean activation to represent a category. Concretely, let $\mathbf{s}_y \in \mathcal{A}_y \cup \bar{\mathbf{a}}_y$ be a statistic for category $y$. Let $S_{\text{large}}$ denote a statistic set $\{\mathbf{s}_1, ..., \mathbf{s}_{|\mathcal{C}_{\text{large}}|}\}$ with one for each category in $\mathcal{C}_{\text{large}}$. We sample activations $\mathbf{s}_y$ for each category $y$ from $\mathcal{A}_y \cup \bar{\mathbf{a}}_y$ with a probability $p_{\text{mean}}$ to use $\bar{\mathbf{a}}_y$ and $1 - p_{\text{mean}}$ to sample uniformly from $\mathcal{A}_y$. Now, we learn $\phi$ to minimize the loss defined by

$$\mathcal{L}(\phi) = \sum_{(y,x) \in \mathcal{D}_{\text{large}}} \mathbb{E}_{S_{\text{large}}} \left[ -\phi\left(\mathbf{s}_y\right) \mathbf{a}(x) + \log \sum_{y' \in \mathcal{C}_{\text{large}}} e^{\phi\left(\mathbf{s}_{y'}\right)\mathbf{a}(x)} \right] + \lambda ||\phi|| \tag{2.2}$$

### 2.2.2 Inference

During inference we include $\mathcal{C}_{\text{few}}$, which calls for a statistic set for all categories $S = \{\mathbf{s}_1, ..., \mathbf{s}_{|\mathcal{C}|}\}$, where $\mathcal{C} = \mathcal{C}_{\text{large}} \cup \mathcal{C}_{\text{few}}$. Each statistic set $S$ can generate a set of parameters $\{\phi(\mathbf{s}_1), ..., \phi(\mathbf{s}_{|\mathcal{C}|})\}$ that can be used for building a classifier on $\mathcal{C}$. Since we have more than one possible set $S$ from the dataset $\mathcal{D} = \mathcal{D}_{\text{large}} \cup \mathcal{D}_{\text{few}}$, we can do classification based on all the possible $S$.

Formally, we compute the probability of $x$ being in category $y$ by

$$P(y|x) = e^{\mathbb{E}_S[\phi(\mathbf{s}_y)\mathbf{a}(x)]} \Big/ \left( \sum_{y' \in \mathcal{C}} e^{\mathbb{E}_S\left[\phi(\mathbf{s}_{y'})\mathbf{a}(x)\right]} \right) \tag{2.3}$$

However, classifying images with the above equation is time-consuming since it computes the expectations over the entire space of $S$ which is exponentially large. We show in the following that if we assume $\phi$ to be linear, then this expectation can be computed efficiently.

In the linear case, $\phi$ is a matrix $\Phi$. The predicted parameter for category $y$ is

$$\hat{\mathbf{w}}_y = \Phi \cdot \mathbf{s}_y \tag{2.4}$$

The inner product of $x$ before the softmax function for category $y$ is

$$h(\mathbf{s}_y, \mathbf{a}(x)) = \hat{\mathbf{w}}_y \cdot \mathbf{a}(x) = \Phi \cdot \mathbf{s}_y \cdot \mathbf{a}(x) \tag{2.5}$$

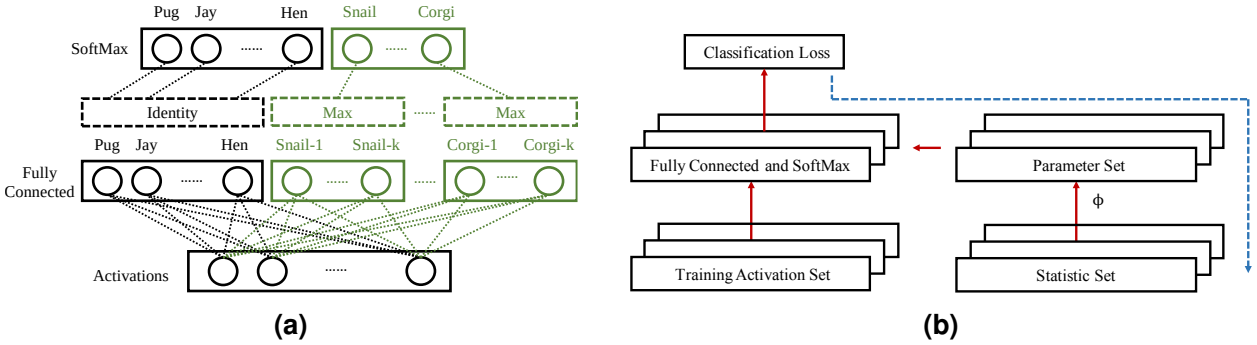



**Figure 2-4.** Illustration of the novel category adaption (a) and the training strategies for parameter predictor $\phi$ (b). (b): red and solid arrows show the feedforward data flow, while blue and dashed arrow shows the backward gradient flow.

If $\mathbf{a}(x)$ and $\mathbf{s}_y$ are normalized, then by setting $\Phi$ as the identity matrix, $h(\mathbf{s}_y, \mathbf{a}(x))$ is equivalent to the cosine similarity between $\mathbf{s}_y$ and $\mathbf{a}(x)$. Essentially, by learning $\Phi$, we are learning a more general similarity metric on the activations $\mathbf{a}(x)$ by capturing correlations between different dimensions of the activations. We will show more comparisons between the learned $\Phi$ and identity matrix in §2.4.1.5. Because of the linearity of $\phi$, the probability of $x$ being in category $y$ simplifies to

$$P(y|x) = e^{\mathbf{a}(x)\cdot\phi(\mathbb{E}_S[\mathbf{s}_y])} \Big/ \left(\sum_{y'\in\mathcal{C}} e^{\mathbf{a}(x)\cdot\phi(\mathbb{E}_S[\mathbf{s}_{y'}])}\right) = \quad e^{\mathbf{a}(x)\cdot\Phi\cdot\mathbb{E}_S[\mathbf{s}_y]} \Big/ \left(\sum_{y'\in\mathcal{C}} e^{\mathbf{a}(x)\cdot\Phi\cdot\mathbb{E}_S[\mathbf{s}_{y'}]}\right) \tag{2.6}$$

Now $\mathbb{E}_S[\mathbf{s}_y]$ can be pre-computed which is efficient. Adapting to novel categories only requires updating the corresponding $\mathbb{E}_S[\mathbf{s}_y]$. Although it is ideal to keep the linearity of $\phi$ to reduce the amount of computation, introducing non-linearity could potentially improve the performance. To keep the efficiency, we still push in the expectation and approximate Eq. 2.3 as in Eq. 2.6.

When adding categories $y \in \mathcal{C}_{\text{few}}$, the estimate of $\mathbb{E}_S[\mathbf{s}_y]$ may not be reliable since the number of samples is small. Besides, Eq. 2.2 models the sampling from one-shot and mean activations. Therefore, we take a mixed strategy for parameter prediction, *i.e.*, we use $\mathbb{E}_S[\mathbf{s}_y]$ to predict parameters for category $y \in \mathcal{C}_{\text{large}}$, but for $\mathcal{C}_{\text{few}}$ we treat each sample as a newly added category, as shown in Figure 2-4a. For each novel category in $C_{\text{few}}$, we compute the maximal response of the activation of the test image to the parameter set predicted from each activation in the statistic set of the corresponding novel category in $C_{\text{few}}$. We use them as the inputs to

the SoftMax layer to compute the probabilities.

### 2.2.3 Training Strategy

The objective of training is to find $\phi$ that minimizes Eq. 2.2. There are many methods to do this. We approach this by using stochastic gradient descent with weight decay and momentum. Figure 2-4b demonstrates the training strategy of the parameter predictor $\phi$. We train $\phi$ on $\mathcal{D}_{\text{large}}$ with categories $\mathcal{C}_{\text{large}}$. For each batch of the training data, we sample $|\mathcal{C}_{\text{large}}|$ statistics $\mathbf{s}_y$ from $\mathcal{A}_y \cup \bar{\mathbf{a}}_y$ to build a statistic set $S$ with one for each category $y$ in $\mathcal{C}_{\text{large}}$. Next, we sample a training activation set $T$ from $\mathcal{D}_{\text{large}}$ with one for each category in $\mathcal{C}_{\text{large}}$. In total, we sample $2|\mathcal{C}_{\text{large}}|$ activations. The activations in the statistic sets are fed to $\phi$ to generate parameters for the fully connected layer. With the predicted parameters for each category in $\mathcal{C}_{\text{large}}$, the training activation set then is used to evaluate their effectiveness by classifying the training activations. At last, we compute the classification loss with respect to the ground truth, based on which we calculate the gradients and back-propagate them in the path shown in Figure 2-4b. After the gradient flow passes through $\phi$, we update $\phi$ according to the gradients.

### 2.2.4 Implementation Details

#### 2.2.4.1 Full ImageNet Dataset

Our major experiments are conducted on ILSVRC 2015 [230]. ILSVRC 2015 is a large-scale image dataset with $1000$ categories, each of which has about $1300$ images for training, and $50$ images for validation. For the purpose of studying both the large-scale and the few-shot settings at the same time, ILSVRC 2015 is split to two sets by the categories. The training data from $900$ categories are collected into $\mathcal{D}_{\text{large}}$, while the rest $100$ categories are gathered as $\mathcal{D}_{\text{few}}$.

We first train a $50$-layer ResNet [108] on $\mathcal{D}_{\text{large}}$. We use the outputs of the global average pooling layer as the activation $\mathbf{a}(x)$ of an image $x$. For efficiency, we compute the activation $\mathbf{a}(x)$ for each image $x$ before the experiments as well as the mean activations $\bar{\mathbf{a}}_y$. Following the training strategy shown in §2.2.3, for each batch, we sample $900$ activations as the statistic

set and $900$ activations as the training activation set. We compute the parameters using the statistic set, and copy the parameters into the fully connected layer. Then, we feed the training activations into the fully connected layer, calculate the loss and back-propagate the gradients. Next, we redirect the gradient flow into $\phi$. Finally, we update $\phi$ using stochastic gradient descent. The learning rate is set to $0.001$. The weight decay is set to $0.0005$ and the momentum is set to $0.9$. We train $\phi$ on $\mathcal{D}_{\text{large}}$ for $300$ epochs, each of which has $250$ batches. $p_{\text{mean}}$ is set to $0.9$.

For the parameter predictor, we implement three different $\phi$: $\phi^1$, $\phi^2$ and $\phi^{2*}$. $\phi^1$ is a one-layer fully connected model. $\phi^2$ is defined as a sequential network with two fully connected layers in which each maps from $2048$ dimensional features to $2048$ dimensional features and the first one is followed by a ReLU non-linearity layer [196]. The final outputs are normalized to unity in order to speed up training and ensure generalizability. By introducing non-linearity, we observe slight improvements on the accuracies for both $\mathcal{C}_{\text{large}}$ and $\mathcal{C}_{\text{few}}$. To demonstrate the effect of minimizing Eq. 2.2 instead of Eq. 2.1, we train another $\phi^{2*}$ which has the same architecture as $\phi^2$ but minimizes Eq. 2.1. As we will show later through experiments, $\phi^{2*}$ has a strong bias towards $\mathcal{C}_{\text{large}}$.

#### 2.2.4.2 MiniImageNet Dataset

For comparison purposes, we also test our method on MiniImageNet dataset [260], a simplified subset of ILSVRC 2015. This dataset has $80$ categories for $D_{\text{large}}$ and $20$ categories for $D_{\text{few}}$. Each category has $600$ images. Each image is of size $84 \times 84$. For the fairness of comparisons, we train two convolutional neural networks to get the activation functions $\mathbf{a}(\cdot)$. The first one is the same as that of Matching Network [260], and the second one is a wide residual network [305]. We train the wide residual network WRN-$28$-$10$ [305] on $D_{\text{large}}$, following its configuration for CIFAR-100 dataset [141]. There are some minor modifications to the network architecture as the input size is different. To follow the architecture, the input size is set to $80 \times 80$. The images will be rescaled to this size before training and evaluation. There will be $3$ times of downsampling rather than $2$ times as for CIFAR dataset. The training process follows WRN-$28$-$10$ [305]. We

also use the output of the global average pooling layer as the activation $\mathbf{a}(x)$ of an image $x$. For the parameter predictor $\phi$, we train it by following the settings of $\phi^2$ for the full ImageNet dataset except that now the dimension corresponds to the output of the activations of the convolutional neural networks.

## 2.3 Related Work

### 2.3.1 Large-Scale Image Recognition

We have witnessed an evolution of image datasets over the last few decades. The sizes of the early datasets are relatively small. Each dataset usually collects images on the order of tens of thousands. Representative datasets include Caltech-101 [74], Caltech-256 [97], Pascal VOC [70], and CIFAR-10/100 [141]. Nowadays, large-scale datasets are available with millions of detailed image annotations, *e.g.* ImageNet [230] and MS COCO [173]. With datasets of this scale, machine learning methods that have large capacity start to prosper, and the most successful ones are convolutional neural network based [108], [120], [142], [239], [269].

### 2.3.2 Few-Shot Image Recognition

Unlike large-scale image recognition, the research on few-shot learning has received limited attention from the community due to its inherent difficulty, thus is still at an early stage of development. As an early attempt, Fei-Fei *et al.* proposed a variational Bayesian framework for one-shot image classification [74]. A method called Hierarchical Bayesian Program Learning [146] was later proposed to specifically approach the one-shot problem on character recognition by a generative model. On the same character recognition task, Koch *et al.* developed a siamese convolutional network [137] to learn the representation from the dataset and modeled the few-shot learning as a verification task. Later, Matching Network [260] was proposed to approach the few-shot learning task by modeling the problem as a $k$-way $m$-shot image retrieval problem using attention and memory models. Following this work, Ravi and Larochelle proposed an

LSTM-based meta-learner optimizer [224], and Chelsea *et al.* proposed a model-agnostic meta-learning method [77]. Although they show state-of-the-art performances on their few-shot learning tasks, they are not flexible for both large-scale and few-shot learning since $k$ and $m$ are fixed in their architectures. We will compare ours with these methods on their tasks for fair comparisons.

### 2.3.3 Unified Approach

Learning a metric then using nearest neighbor [137], [174], [267] is applicable but not necessarily optimal to the unified problem of large-scale and few-shot learning since it is possible to train a better model on the large-scale part of the dataset using the methods in §2.3.1. Mao *et al.* proposed a method called Learning like a Child [189] specifically for fast novel visual concept learning using hundreds of examples per category while keeping the original performance. However, this method is less effective when the training examples are extremely insufficient, *e.g.* $< 6$ in this chapter.

## 2.4 Results

### 2.4.1 Full ImageNet Classification

In this section we describe our experiments and compare our approach with other strong baseline methods. As stated in §2.1, there are three aspects to consider in evaluating a method: (1) its performance on the few-shot set $\mathcal{D}_{\text{few}}$, (2) its performance on the large-scale set $\mathcal{D}_{\text{large}}$, and (3) its computation overhead of adding novel categories and the complexity of image inference. In the following paragraphs, we will cover the settings of the baseline methods, compare the performances on the large-scale and the few-shot sets, and discuss their efficiencies.

#### 2.4.1.1 Baseline Methods

The baseline methods must be applicable to both large-scale and few-shot learning settings. We compare our method with a fine-tuned $50$-layer ResNet [108], Learning like a Child [189] with a pre-trained $50$-layer ResNet as the starting network, Siamese-Triplet Network [137], [174] using three $50$-layer ResNets with shared parameters, and the nearest neighbor using the pre-trained $50$-layer ResNet convolutional features. We will elaborate individually on how to train and use them.

As mentioned in §2.2.4, we first train a $900$-category classifier on $\mathcal{D}_{\text{large}}$. We will build other baseline methods using this classifier as the starting point. For convenience, we denote this classifier as $\mathcal{R}^{\text{pt}}_{\text{large}}$, where pt stands for "pre-trained". Next, we add the novel categories $\mathcal{C}_{\text{few}}$ to each method. For the $50$-layer ResNet, we fine tune $\mathcal{R}^{\text{pt}}_{\text{large}}$ with the newly added images by extending the fully connected layer to generate $1000$ classification outputs. Note that we will limit the number of training samples of $\mathcal{C}_{\text{few}}$ for the few-shot setting. For Learning like a Child, however, we fix the layers before the global average pooling layer, extend the fully connected layer to include $1000$ classes, and only update the parameters for $\mathcal{C}_{\text{few}}$ in the last classification layer. Since we have full access to $\mathcal{D}_{\text{large}}$, we do not need Baseline Probability Fixation [189]. The nearest neighbor with cosine distance can be directly used for both tasks given the pre-trained deep features.

The other method we compare is Siamese-Triplet Network [137], [174]. Siamese network is proposed to approach the few-shot learning problem on Omniglot dataset [145]. In our experiments, we find that its variant Triplet Network [174], [267] is more effective since it learns feature representation from relative distances between positive and negative pairs instead of directly doing binary classification from the feature distance. Therefore, we use the Triplet Network from [174] on the few-shot learning problem, and upgrade its body net to the pre-trained $\mathcal{R}^{\text{pt}}_{\text{large}}$. We use cosine distance as the distance metric and fine-tune the Triplet Network. For inference, we use nearest neighbor with cosine distance. We use some techniques to improve

| Method | $\mathcal{D}_{\text{large}}$ | $\mathcal{D}_{\text{few}}$ | FT | Top-1 $\mathcal{C}_{\text{large}}$ | Top-5 $\mathcal{C}_{\text{large}}$ | Top-1 $\mathcal{C}_{\text{few}}$ | Top-5 $\mathcal{C}_{\text{few}}$ |
|---|---|---|---|---|---|---|---|
| NN + Cosine | 100% | 1 | N | 71.54% | 91.20% | 1.72% | 5.86% |
| NN + Cosine | 10% | 1 | N | 67.68% | 88.90% | 4.42% | 13.36% |
| NN + Cosine | 1% | 1 | N | 61.11% | 85.11% | 10.42% | 25.88% |
| Triplet Network [137], [174] | 100% | 1 | N | 70.47% | 90.61% | 1.26% | 4.94% |
| Triplet Network [137], [174] | 10% | 1 | N | 66.64% | 88.42% | 3.48% | 11.40% |
| Triplet Network [137], [174] | 1% | 1 | N | 60.09% | 84.83% | 8.84% | 22.24% |
| Fine-Tuned ResNet [108] | 100% | 1 | Y | 76.28% | 93.17% | 2.82% | 13.30% |
| Learning like a Child [189] | 100% | 1 | Y | 76.71% | 93.24% | 2.90% | 17.14% |
| Ours-$\phi^1$ | 100% | 1 | N | 72.56% | 91.12% | **19.88%** | **43.20%** |
| Ours-$\phi^2$ | 100% | 1 | N | 74.17% | 91.79% | **21.58%** | **45.82%** |
| Ours-$\phi^{2*}$ | 100% | 1 | N | 75.63% | 92.92% | 14.32% | 33.84% |
| NN + Cosine | 100% | 2 | N | 71.54% | 91.20% | 3.34% | 9.88% |
| NN + Cosine | 10% | 2 | N | 67.66% | 88.89% | 7.60% | 19.94% |
| NN + Cosine | 1% | 2 | N | 61.04% | 85.04% | 15.14% | 35.70% |
| Triplet Network [137], [174] | 100% | 2 | N | 70.47% | 90.61% | 2.34% | 8.30% |
| Triplet Network [137], [174] | 10% | 2 | N | 66.63% | 88.41% | 6.10% | 17.46% |
| Triplet Network [137], [174] | 1% | 2 | N | 60.02% | 84.74% | 13.42% | 32.38% |
| Fine-Tuned ResNet [108] | 100% | 2 | Y | 76.27% | 93.13% | 10.32% | 30.34% |
| Learning like a Child [189] | 100% | 2 | Y | 76.68% | 93.17% | 11.54% | 37.68% |
| Ours-$\phi^1$ | 100% | 2 | N | 71.94% | 90.62% | **25.54%** | **52.98%** |
| Ours-$\phi^2$ | 100% | 2 | N | 73.43% | 91.13% | **27.44%** | **55.86%** |
| Ours-$\phi^{2*}$ | 100% | 2 | N | 75.44% | 92.74% | 18.70% | 43.92% |
| NN + Cosine | 100% | 3 | N | 71.54% | 91.20% | 4.58% | 12.72% |
| NN + Cosine | 10% | 3 | N | 67.65% | 88.88% | 9.86% | 24.96% |
| NN + Cosine | 1% | 3 | N | 60.97% | 84.95% | 18.68% | 42.16% |
| Triplet Network [137], [174] | 100% | 3 | N | 70.47% | 90.61% | 3.22% | 11.48% |
| Triplet Network [137], [174] | 10% | 3 | N | 66.62% | 88.40% | 8.52% | 22.52% |
| Triplet Network [137], [174] | 1% | 3 | N | 59.97% | 84.66% | 17.08% | 38.06% |
| Fine-Tuned ResNet [108] | 100% | 3 | Y | 76.25% | 93.07% | 16.76% | 39.92% |
| Learning like a Child [189] | 100% | 3 | Y | 76.55% | 93.00% | 18.56% | 50.70% |
| Ours-$\phi^1$ | 100% | 3 | N | 71.56% | 90.21% | **28.72%** | **58.50%** |
| Ours-$\phi^2$ | 100% | 3 | N | 72.98% | 90.59% | **31.20%** | **61.44%** |
| Ours-$\phi^{2*}$ | 100% | 3 | N | 75.34% | 92.60% | 22.32% | 49.76% |

**Table 2-I.** Comparing 1000-way accuracies with feature extractor $\mathbf{a}(\cdot)$ pre-trained on $\mathcal{D}_{\text{large}}$. For different $\mathcal{D}_{\text{few}}$ settings, red: the best few-shot accuracy, and blue: the second best.

the speed, which will be discussed later in the efficiency analysis.

#### 2.4.1.2 Few-Shot Accuracy

We first investigate the few-shot learning setting where we only have several training examples for $\mathcal{C}_{\text{few}}$. Specifically, we study the performances of different methods when $\mathcal{D}_{\text{few}}$ has for each category 1, 2, and 3 samples. It is worth noting that our task is much harder than the previously studied few-shot learning: we are evaluating the top predictions out of $1000$ candidate categories, *i.e.*, $1000$-way accuracies while previous work is mostly interested in $5$-way or $20$-way accuracies [77], [137], [174], [224], [260].

With the pre-trained $\mathcal{R}^{\text{pt}}_{\text{large}}$, the training samples in $\mathcal{D}_{\text{few}}$ are like invaders to the activation space for $\mathcal{C}_{\text{large}}$. Intuitively, there will be a trade-off between the performances on $\mathcal{C}_{\text{large}}$ and $\mathcal{C}_{\text{few}}$. This is true, especially for non-parametric methods. Table 2-I shows the performances on the validation set of ILSVRC 2015 [230]. The second column is the percentage of data of $\mathcal{D}_{\text{large}}$ in use, and the third column is the number of samples used for each category in $\mathcal{D}_{\text{few}}$. Note that fine-tuned ResNet [108] and Learning like a Child [189] require fine-tuning while others do not.

Triplet Network is designed to do few-shot image inference by learning feature representations that adapt to the chosen distance metric. It has better performance on $\mathcal{C}_{\text{few}}$ compared with the fine-tuned ResNet and Learning like a Child when the percentage of $\mathcal{D}_{\text{large}}$ in use is low. However, its accuracies on $\mathcal{C}_{\text{large}}$ are sacrificed a lot in order to favor few-shot accuracies. We also note that if full category supervision is provided, the activations of training a classifier do better than that of training a Triplet Network. We speculate that this is due to the less supervision of training a Triplet Network which uses losses based on fixed distance preferences. Fine-tuning and Learning like a Child are training-based, thus are able to keep the high accuracies on $\mathcal{D}_{\text{large}}$, but perform badly on $\mathcal{D}_{\text{few}}$ which does not have sufficient data for training. Compared with them, our method shows state-of-the-art accuracies on $\mathcal{C}_{\text{few}}$ without compromising too much the performances on $\mathcal{C}_{\text{large}}$.

Table 2-I also compares $\phi^2$ and $\phi^{2*}$, which are trained to minimize Eq. 2.2 and Eq. 2.1,

| Classifier | Top-1 $\mathcal{C}_{\text{large}}$ | Top-5 $\mathcal{C}_{\text{large}}$ | Top-1 $\mathcal{C}_{\text{few}}$ | Top-5 $\mathcal{C}_{\text{few}}$ |
|---|---|---|---|---|
| NN | 70.25% | 89.98% | 52.46% | 80.94 |
| Linear | 75.20% | 92.38% | 60.50% | 87.58 |

**Table 2-II.** Oracle 1000-way accuracies of the feature extractor $\mathbf{a}(\cdot)$ pre-trained on $\mathcal{D}_{\text{large}}$.

respectively. Since during training $\phi^{2*}$ only mean activations are sampled, it shows a bias towards $\mathcal{C}_{\text{large}}$. However, it still outperforms other baseline methods on $\mathcal{C}_{\text{few}}$. In short, modeling using Eq. 2.2 and Eq. 2.1 shows a tradeoff between $\mathcal{C}_{\text{large}}$ and $\mathcal{C}_{\text{few}}$.

#### 2.4.1.3 Oracles

Here we explore the upper bound performance on $\mathcal{C}_{\text{few}}$. In this setting we have all the training data for $\mathcal{C}_{\text{large}}$ and $\mathcal{C}_{\text{few}}$ in ImageNet. For the fixed feature extractor $\mathbf{a}(\cdot)$ pre-trained on $\mathcal{D}_{\text{large}}$, we can train a linear classifier on $\mathcal{C}_{\text{large}}$ and $\mathcal{C}_{\text{few}}$, or use nearest neighbor, to see what are the upper bounds of the pre-trained $\mathbf{a}(\cdot)$. Table 2-II shows the results. The performances are evaluated on the validation set of ILSVRC 2015 [230] which has $50$ images for each category. The feature extractor pre-trained on $\mathcal{D}_{\text{large}}$ demonstrates reasonable accuracies on $\mathcal{C}_{\text{few}}$ which it has never seen during training for both parametric and non-parametric methods.

#### 2.4.1.4 Efficiency Analysis

We briefly discuss the efficiencies of each method including ours on the adaptation to novel categories and the image inference. The methods are tested on NVIDIA Tesla K40M GPUs. For adapting to novel categories, fine-tuned ResNet and Learning like a Child require re-training the neural networks. For re-training one epoch of the data, fine-tuned ResNet and Learning like a Child both take about 1.8 hours on 4 GPUs. Our method only needs to predict the parameters for the novel categories using $\phi$ and add them to the original neural network. This process takes 0.683s using one GPU for adapting the network to $100$ novel categories with one example each. Siamese-Triplet Network and nearest neighbor with cosine distance require no operations for adapting to novel categories as they are ready for feature extraction.

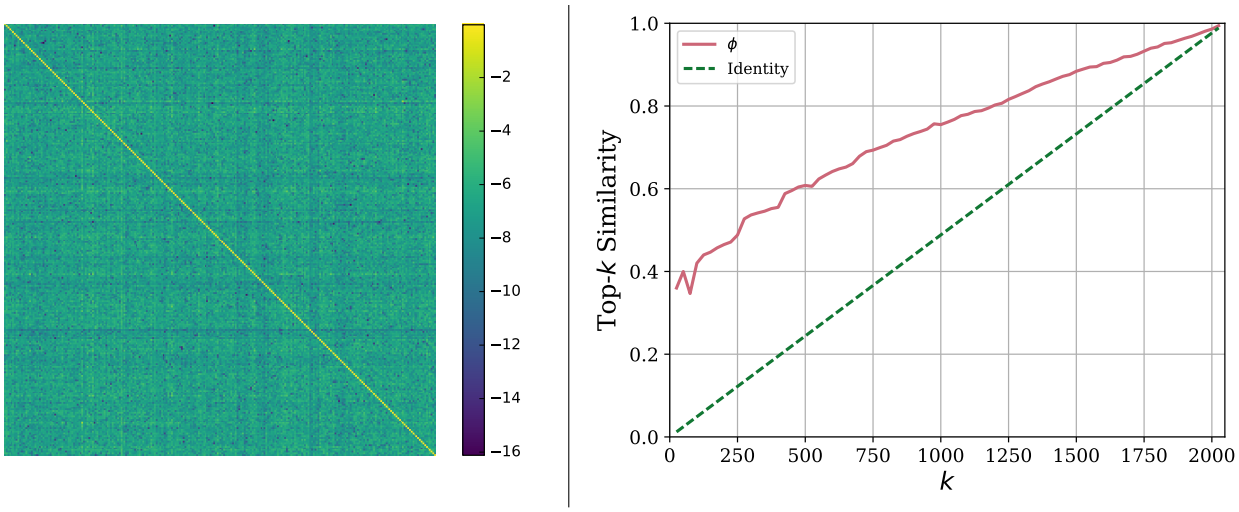



**Figure 2-5.** Visualization of the upper-left $256 \times 256$ submatrix of $\phi^1$ in $\log$ scale (left) and top-$k$ similarity between $\phi^1$, $\mathbb{1}$ and $\mathbf{w}^{\text{pt}}_{\text{large}}$ (right). In the right plotting, red and solid lines are similarities between $\phi^1$ and $\mathbf{w}^{\text{pt}}_{\text{large}}$, and green and dashed lines are between $\mathbb{1}$ and $\mathbf{w}^{\text{pt}}_{\text{large}}$.

For image inference, Siamese-Triplet Network and nearest neighbor are very slow since they will look over the entire dataset. Without any optimization, this can take 2.3 hours per image when we use the entire $D_{\text{large}}$. To speed up this process in order to do a comparison with ours, we first pre-compute all the features. Then, we use a deep learning framework to accelerate the cosine similarity computation. At the cost of 45GB memory usage and the time for feature pre-computation, we manage to lower the inference time of them to 37.867ms per image. Fine-tuned ResNet, Learning like a Child and our method are very fast since, at the inference stage, these three methods are just normal deep neural networks. The inference speed of these methods is about 6.83ms per image on one GPU when the batch size is set to 32. In a word, compared with other methods, our method is fast and efficient in both the novel category adaptation and the image inference.

#### 2.4.1.5 Comparing Activation Impacts

In this subsection we investigate what $\phi^1$ has learned that helps it perform better than the cosine distance, which is a special solution for one-layer $\phi$ by setting $\phi$ to the identity matrix $\mathbb{1}$. We first visualize the matrix $\phi^1_{ij}$ in $\log$ scale as shown in the left image of Figure 2-5. Due to the

space limit, we only show the upper-left $256 \times 256$ submatrix. Not surprisingly, the values on the diagonal dominate the matrix. We observe that along the diagonal, the maximum is $0.976$ and the minimum is $0.744$, suggesting that different from $\mathbb{1}$, $\phi^1$ does not use each activation channel equally. We speculate that this is because the pre-trained activation channels have different distributions of magnitudes and different correlations with the classification task. These factors can be learned by the last fully connected layer of $\mathcal{R}^{\text{pt}}_{\text{large}}$ with large amounts of data but are assumed equal for every channel in cosine distance. This motivates us to investigate the impact of each channel of the activation space.

For a fixed activation space, we define the *impact* of its $j$-th channel on mapping $\phi$ by $I_j(\phi) = \sum_i |\phi_{ij}|$. Similarly, we define the activation impact $I_j(\cdot)$ on $\mathbf{w}^{\text{pt}}_{\text{large}}$ which is the parameter matrix of the last fully connected layer of $\mathcal{R}^{\text{pt}}_{\text{large}}$. For cosine distance, $I_j(\mathbb{1}) = 1, \forall j$. Intuitively, we are evaluating the impact of each channel of $\mathbf{a}$ on the output by adding all the weights connected to it. For $\mathbf{w}^{\text{pt}}_{\text{large}}$ which is trained for the classification task using large amounts of data, if we normalize $I(\mathbf{w}^{\text{pt}}_{\text{large}})$ to unity, the mean of $I(\mathbf{w}^{\text{pt}}_{\text{large}})$ over all channel $j$ is $2.13\text{e-}2$ and the standard deviation is $5.83\text{e-}3$. $\mathbf{w}^{\text{pt}}_{\text{large}}$ does not use channels equally, either.

In fact, $\phi^1$ has a high similarity with $\mathbf{w}^{\text{pt}}_{\text{large}}$. We show this by comparing the orders of the channels sorted by their impacts. Let top-$k(S)$ find the indexes of the top-$k$ elements of $S$. We define the top-$k$ similarity of $I(\phi)$ and $I(\mathbf{w}^{\text{pt}}_{\text{large}})$ by

$$\text{OS}(\phi, \mathbf{w}^{\text{pt}}_{\text{large}}, k) = \mathbf{card}\left(\text{top-}k(I(\phi)) \cap \text{top-}k(I(\mathbf{w}^{\text{pt}}_{\text{large}}))\right)/k \tag{2.7}$$

where $\mathbf{card}$ is the cardinality of the set. The right image of Figure 2-5 plots the two similarities, from which we observe high similarity between $\phi$ and $\mathbf{w}^{\text{pt}}_{\text{large}}$ compared to the random order of $\mathbb{1}$. From this point of view, $\phi^1$ outperforms the cosine distance due to its better usage of the activations.

### 2.4.2 MiniImageNet Classification

In this subsection we compare our method with the previous state-of-the-arts on the MiniImageNet dataset. Unlike ImageNet classification, the task of MiniImageNet is to find the correct

| Method | 1-Shot | 5-Shot |
|---|---|---|
| Fine-Tuned Baseline | 28.86 $\pm$ 0.54% | 49.79 $\pm$ 0.79% |
| Nearest Neighbor | 41.08 $\pm$ 0.70% | 51.04 $\pm$ 0.65% |
| Matching Network [260] | 43.56 $\pm$ 0.84% | 55.31 $\pm$ 0.73% |
| Meta-Learner LSTM [224] | 43.44 $\pm$ 0.77% | 60.60 $\pm$ 0.71% |
| MAML [77] | 48.70 $\pm$ 1.84% | 63.11 $\pm$ 0.92% |
| Ours-Simple | **54.53** $\pm$ 0.40% | **67.87** $\pm$ 0.20% |
| Ours-WRN | **59.60** $\pm$ 0.41% | **73.74** $\pm$ 0.19% |

**Table 2-III.** $5$-way accuracies on MiniImageNet with $95\%$ confidence interval. Red: the best, and blue: the second best.

category from $5$ candidates, each of which has $1$ example or $5$ examples for reference. The methods are only evaluated on $D_{\text{few}}$, which has $20$ categories. For each task, we uniformly sample $5$ categories from $D_{\text{few}}$. For each of the categories, we randomly select one or five images as the references, depending on the settings, then regard the rest images of the $5$ categories as the test images. For each task, we will have an average accuracy over these $5$ categories. We repeat the task with different categories and report the mean of the accuracies with the $95\%$ confidence interval.

Table 2-III summarizes the few-shot accuracies of our method and the previous state-of-the-arts. For fair comparisons, we implement two convolutional neural networks. The convolutional network of Ours-Simple is the same as that of Matching Network [260] while Ours-WRN uses WRN-28-10 [305] as stated in §2.2.4. The experimental results demonstrate that our average accuracies are better than the previous state-of-the-arts by a large margin for both the Simple and WRN implementations.

It is worth noting that the methods [77], [224], [260] are not evaluated in the full ImageNet classification task. This is because the architectures of these methods, following the problem formulation of Matching Network [260], can only deal with the test tasks that are of the same number of reference categories and images as that of the training tasks, limiting their flexibilities for classification tasks of arbitrary number of categories and reference images. In contrast, our proposed method has no assumptions regarding the number of the reference categories and

the images, while achieving good results on both tasks. From this perspective, our methods are better than the previous state-of-the-arts in terms of both performance and flexibility.

## 2.5 Conclusion

In this chapter, we study a novel problem: can we develop a unified approach that works for both large-scale and few-shot learning. Our motivation is based on the observation that in the final classification layer of a pre-trained neural network, the parameter vector and the activation vector have highly similar structures in space. This motivates us to learn a category-agnostic mapping from activations to parameters. Once this mapping is learned, the parameters for any novel category can be predicted by a simple forward pass, which is significantly more convenient than re-training used in parametric methods or enumeration of training set used in non-parametric approaches.

We experiment with our novel approach on the MiniImageNet dataset and the challenging full ImageNet dataset. The challenges of the few-shot learning on the full ImageNet dataset are from the large number of categories (1000) and the very limited number ($< 4$) of training samples for $\mathcal{C}_{\text{few}}$. On the full ImageNet dataset, we show promising results, achieving state-of-the-art classification accuracy on novel categories by a significant margin while maintaining comparable performance on the large-scale classes. On the small MiniImageNet dataset, we also outperform the previous state-of-the-art methods by a large margin. The experimental results demonstrate the effectiveness of the proposed method for learning a category-agnostic mapping.

# Chapter 3

# Deep Co-Training for Semi-Supervised Image Recognition

In this chapter, we study the problem of semi-supervised image recognition, which is to learn classifiers using both labeled and unlabeled images. We present Deep Co-Training, a deep learning based method inspired by the Co-Training framework. The original Co-Training learns two classifiers on two *views* which are data from *different* sources that describe *the same* instances. To extend this concept to deep learning, Deep Co-Training trains multiple deep neural networks to be the different views and exploits adversarial examples to encourage view difference, in order to prevent the networks from collapsing into each other. As a result, the co-trained networks provide different and complementary information about the data, which is necessary for the Co-Training framework to achieve good results. We test our method on SVHN, CIFAR-10/100, and ImageNet datasets, and our method outperforms the previous state-of-the-art methods by a large margin.

## 3.1 Introduction

Deep neural networks achieve the state-of-art performances in many tasks [41], [90], [108], [120], [142], [182], [216], [217], [219], [221], [228], [239], [247], [269], [270], [306], [313]. However, training networks requires large-scale labeled datasets [173], [230] which are usually difficult to collect. Given the massive amounts of unlabeled natural images, the idea to use

datasets without human annotations becomes very appealing [328]. In this chapter, we study the semi-supervised image recognition problem, the task of which is to use the unlabeled images in addition to the labeled images to build better classifiers. Formally, we are provided with an image dataset $\mathcal{D} = \mathcal{S} \cup \mathcal{U}$ where images in $\mathcal{S}$ are labeled and images in $\mathcal{U}$ are not. The task is to build classifiers on the categories $\mathcal{C}$ in $\mathcal{S}$ using the data in $\mathcal{D}$ [57], [144], [231]. The test data contains only the categories that appear in $\mathcal{S}$. The problem of learning models on supervised datasets has been extensively studied, and the state-of-the-art methods are deep convolutional networks [108], [120]. The core problem is how to use the unlabeled $\mathcal{U}$ to help learning on $\mathcal{S}$.

The method proposed in this chapter is inspired by the Co-Training framework [23], which is an award-winning method for semi-supervised learning. It assumes that each data $x$ in $\mathcal{D}$ has two *views*, *i.e.* $x$ is given as $x = (v_1, v_2)$, and each view $v_i$ is sufficient for learning an effective model. For example, the views can have different data sources [23] or different representations [15], [200], [278]. Let $\mathcal{X}$ be the distribution that $\mathcal{D}$ is drawn from. Co-Training assumes that $f_1$ and $f_2$ trained on view $v_1$ and $v_2$ respectively have consistent predictions on $\mathcal{X}$, *i.e.,*

$$f(x) = f_1(v_1) = f_2(v_2), \quad \forall x = (v_1, v_2) \sim \mathcal{X} \qquad \text{(Co-Training Assumption)} \tag{3.1}$$

Based on this assumption, Co-Training proposes a dual-view self-training algorithm: it first learns a separate classifier for each view on $\mathcal{S}$, and then the predictions of the two classifiers on $\mathcal{U}$ are gradually added to $\mathcal{S}$ to continue the training. Blum and Mitchell [23] further show that under an additional assumption that the two views of each instance are conditionally independent given the category, Co-Training has PAC-like guarantees on semi-supervised learning.

Given the superior performances of deep neural networks on supervised image recognition, we are interested in extending the Co-Training framework to apply deep learning to semi-supervised image recognition. A naive implementation is to train two neural networks

simultaneously on $\mathcal{D}$ by modeling Eq. 3.1. But this method suffers from a critical drawback: there is no guarantee that the views provided by the two networks give different and complementary information about each data point. Yet Co-Training is beneficial only if the two views are different, ideally conditionally independent given the category; after all, there is no point in training two identical networks. Moreover, the Co-Training assumption encourages the two models to make similar predictions on both $\mathcal{S}$ and $\mathcal{U}$, which can even lead to collapsed neural networks, as we will show by experiments in Section 3. Therefore, in order to extend the Co-Training framework to take the advantage of deep learning, it is necessary to have a force that pushes networks away to balance the Co-Training assumption that pulls them together.

The force we add to the Co-Training Assumption is *View Difference Constraint* formulated by Eq. 3.2, which encourages the networks to be different

$$\exists \mathcal{X}' : \ f_1(v_1) \neq f_2(v_2), \ \forall x = (v_1, v_2) \sim \mathcal{X}' \quad \text{(View Difference Constraint)} \tag{3.2}$$

The challenge is to find a proper and sufficient $\mathcal{X}'$ that is compatible with Eq. 3.1 (*e.g.* $\mathcal{X}' \cap \mathcal{X} = \varnothing$) and our tasks. We construct $\mathcal{X}'$ by adversarial examples [94].

In this chapter, we present Deep Co-Training (DCT) for semi-supervised image recognition, which extends the Co-Training framework without the drawback discussed above. Specifically, we model the Co-Training assumption by minimizing the expected Jensen-Shannon divergence between the predictions of the two networks on $\mathcal{U}$. To avoid the neural networks from collapsing into each other, we impose the view difference constraint by training each network to be resistant to the adversarial examples [94], [279] of the other. The result of the training is that each network can keep its predictions unaffected on the examples that the other network fails on. In other words, the two networks provide different and complementary information about the data because they are trained not to make errors at the same time on the adversarial examples for them. To summarize, the main contribution of DCT is a differentiable modeling that takes into account both the Co-Training assumption and the view difference constraint. It is an end-to-end solution which minimizes a loss function defined on the dataset $\mathcal{S}$ and $\mathcal{U}$.

Naturally, we extend the dual-view DCT to a scalable multi-view DCT. We test our method on four datasets, SVHN [197], CIFAR10/100 [141] and ImageNet [230], and DCT outperforms the previous state-of-the-arts by a large margin.

## 3.2 Deep Co-Training

In this section, we present our model of Deep Co-Training (DCT) and naturally extend dual-view DCT to multi-view DCT.

### 3.2.1 Co-Training Assumption in DCT

We start with the dual-view case where we are interested in co-training two deep neural networks for image recognition. Following the notations in Section 1, we use $\mathcal{S}$ and $\mathcal{U}$ to denote the labeled and the unlabeled dataset. Let $\mathcal{D} = \mathcal{S} \cup \mathcal{U}$ denote all the provided data. Let $v_1(x)$ and $v_2(x)$ denote the two views of data $x$. In this chapter, $v_1(x)$ and $v_2(x)$ are convolutional representations of $x$ before the final fully-connected layer $f_i(\cdot)$ that classifies $v_i(x)$ to one of the categories in $\mathcal{S}$. On the supervised dataset $\mathcal{S}$, we use the standard cross-entropy loss

$$\mathcal{L}_{\text{sup}}(x, y) = H\Big(y, f_1\big(v_1(x)\big)\Big) + H\Big(y, f_2\big(v_2(x)\big)\Big) \tag{3.3}$$

for any data $(x, y)$ in $\mathcal{S}$ where $y$ is the label for $x$ and $H(p, q)$ is the cross-entropy between distribution $p$ and $q$.

Next, we model the Co-Training assumption. Co-Training assumes that on the distribution $\mathcal{X}$ where $x$ is drawn from, $f_1(v_1(x))$ and $f_2(v_2(x))$ agree on their predictions. In other words, we want networks $p_1(x) = f_1(v_1(x))$ and $p_2(x) = f_2(v_2(x))$ to have close predictions on $\mathcal{U}$. Therefore, we use a natural measure of similarity, the Jensen-Shannon divergence between $p_1(x)$ and $p_2(x)$, *i.e.*,

$$\mathcal{L}_{\text{cot}}(x) = H\Big(\frac{1}{2}\big(p_1(x) + p_2(x)\big)\Big) - \frac{1}{2}\Big(H\big(p_1(x)\big) + H\big(p_2(x)\big)\Big) \tag{3.4}$$

where $x \in \mathcal{U}$ and $H(p)$ is the entropy of $p$. Training neural networks based on the Co-Training assumption minimizes the expected loss $\mathbb{E}[\mathcal{L}_{\text{cot}}]$ on the unlabeled set $\mathcal{U}$. As for the labeled set $\mathcal{S}$, minimizing loss $\mathcal{L}_{\text{sup}}$ already encourages them to have close predictions on $\mathcal{S}$ since they are trained with labels; therefore, minimizing $\mathcal{L}_{\text{cot}}$ on $\mathcal{S}$ is unnecessary, and we only implement it on $\mathcal{U}$ (*i.e.* not on $\mathcal{S}$).

### 3.2.2 View Difference Constraint in DCT

The key condition of Co-Training to be successful is that the two views are different and provide complementary information about each data $x$. But minimizing Eq. 3.3 and 3.4 only encourages the neural networks to output the same predictions on $\mathcal{D} = \mathcal{S} \cup \mathcal{U}$. Therefore, it is necessary to encourage the networks to be different and complementary. To achieve this, we create another set of images $\mathcal{D}'$ where $p_1(x) \neq p_2(x)$, $\forall x \in \mathcal{D}'$, which we will generate by adversarial examples [94], [279].

Since Co-Training assumes that $p_1(x) = p_2(x),\ \forall x \in \mathcal{D}$, we know that $\mathcal{D} \cap \mathcal{D}' = \varnothing$. But $\mathcal{D}$ is all the data we have; therefore, $\mathcal{D}'$ must be built up by a generative method. On the other hand, suppose that $p_1(x)$ and $p_2(x)$ can achieve very high accuracy on naturally obtained data (*e.g.* $\mathcal{D}$), assuming $p_1(x) \neq p_2(x)$, $\forall x \in \mathcal{D}'$ also implies that $\mathcal{D}'$ should be constructed by a generative method.

We consider a simple form of generative method $g(x)$ which takes data $x$ from $\mathcal{D}$ to build $\mathcal{D}'$, *i.e.* $\mathcal{D}' = \{g(x) \mid x \in \mathcal{D}\}$. For any $x \in \mathcal{D}$, we want $g(x) - x$ to be small so that $g(x)$ also looks like a natural image. But when $g(x) - x$ is small, it is very possible that $p_1(g(x)) = p_1(x)$ and $p_2(g(x)) = p_2(x)$. Since Co-Training assumes $p_1(x) = p_2(x)$, $\forall x \in \mathcal{D}$ and we want $p_1(g(x)) \neq p_2(g(x))$, when $p_1(g(x)) = p_1(x)$, it follows that $p_2(g(x)) \neq p_2(x)$. These considerations imply that $g(x)$ is an adversarial example [94] of $p_2$ that fools the network $p_2$ but not the network $p_1$. Therefore, in order to prevent the deep networks from collapsing into each other, we propose to train the network $p_1$ (or $p_2$) to be resistant to the adversarial examples $g_2(x)$ of $p_2$ (or $g_1(x)$ of $p_1$) by minimizing the cross-entropy between $p_2(x)$ and $p_1(g_2(x))$ (or

between $p_1(x)$ and $p_2(g_1(x))$), *i.e.,*

$$\mathcal{L}_{\text{dif}}(x) = H\Big(p_1(x), p_2\big(g_1(x)\big)\Big) + H\Big(p_2(x), p_1\big(g_2(x)\big)\Big) \tag{3.5}$$

Using artificially created examples in image recognition has been studied. They can serve as regularization techniques to smooth outputs [192], or create negative examples to tighten decision boundaries [57], [126]. Now, they are used to make networks different. To summarize the Co-Training with the view difference constraint in a sentence, we want the models to have *the same* predictions on $\mathcal{D}$ but make *different* errors when they are exposed to adversarial attacks. By minimizing Eq. 3.5 on $\mathcal{D}$, we encourage the models to generate complementary representations, each is resistant to the adversarial examples of the other.

### 3.2.3 Training DCT

In Deep Co-Training, the objective function is of the form

$$\mathcal{L} = \mathbb{E}_{(x,y)\in\mathcal{S}}\mathcal{L}_{\text{sup}}(x,y) + \lambda_{\text{cot}}\mathbb{E}_{x\in\mathcal{U}}\mathcal{L}_{\text{cot}}(x) + \lambda_{\text{dif}}\mathbb{E}_{x\in\mathcal{D}}\mathcal{L}_{\text{dif}}(x) \tag{3.6}$$

which linearly combines Eq. 3.3, Eq. 3.4 and Eq. 3.5 with hyperparameters $\lambda_{\text{cot}}$ and $\lambda_{\text{dif}}$. We present one iteration of the training loop in Algorithm 1. The full training procedure repeats the computations in Algorithm 1 for many iterations and epochs using gradient descent with decreasing learning rates.

Note that in each iteration of the training loop of DCT, the two neural networks receive different supervised data. This is to increase the difference between them by providing them with supervised data in different time orders. Consider that the data of the two networks are provided by two data streams $s$ and $\overline{s}$. Each data $d$ from $s$ and $\overline{d}$ from $\overline{s}$ are of the form $[d_s, d_u]$, where $d_s$ and $d_u$ denote a batch of supervised data and unsupervised data, respectively. We call $(s, \overline{s})$ *a bundle of data streams* if their $d_u$ are the same and the sizes of $d_s$ are the same. Algorithm 1 uses a bundle of data streams to provide data to the two networks. The idea of using bundles of data streams is important for scalable multi-view Deep Co-Training, which we will present in the following subsections.

**Algorithm 1:** One Iteration of the Training Loop of Deep Co-Training

1 **Data Sampling** Sample data batch $b_1 = (x_{b_1}, y_{b_1})$ for $p_1$ and $b_2 = (x_{b_2}, y_{b_2})$ for $p_2$ from $\mathcal{S}$ s.t. $|b_1| = |b_2| = b$. Sample data batch $b_u = (x_u)$ from $\mathcal{U}$.

2 **Create Adversarial Examples** Compute the adversarial examples $g_1(x)$ of $p_1$ for all $x \in x_{b_1} \cup x_u$ and $g_2(x)$ of $p_2$ for all $x \in x_{b_2} \cup x_u$ using FGSM [94].

3 $\mathcal{L}_{\text{sup}} = \frac{1}{b}\Big[\sum_{(x,y)\in b_1} H(y, p_1(x)) + \sum_{(x,y)\in b_2} H(y, p_2(x))\Big]$

4 $\mathcal{L}_{\text{cot}} = \frac{1}{|b_u|}\sum_{x\in b_u}\Big[H\Big(\frac{1}{2}\big(p_1(x) + p_2(x)\big)\Big) - \frac{1}{2}\Big(H\big(p_1(x)\big) + H\big(p_2(x)\big)\Big)\Big]$

5 $\mathcal{L}_{\text{dif}} = \frac{1}{b + |b_u|}\Big[\sum_{x\in x_1\cup x_u} H(p_1(x), p_2(g_1(x))) + \sum_{x\in x_2\cup x_u} H(p_2(x), p_1(g_2(x)))\Big]$

6 $\mathcal{L} = \mathcal{L}_{\text{sup}} + \lambda_{\text{cot}}\mathcal{L}_{\text{cot}} + \lambda_{\text{dif}}\mathcal{L}_{\text{dif}}$

7 **Update** Compute the gradients with respect to $\mathcal{L}$ by backpropagation and update the parameters of $p_1$ and $p_2$ using gradient descent.

### 3.2.4 Multi-View DCT

In the previous subsection, we introduced our model of dual-view Deep Co-Training. But dual-view is only a special case of multi-view learning, and multi-view co-training has also been studied for other problems [284], [324]. In this subsection, we present a scalable method for multi-view Deep Co-Training. Here, the scalability means that the hyperparameters $\lambda_{\text{cot}}$ and $\lambda_{\text{dif}}$ in Eq. 3.6 that work for dual-view DCT are also suitable for increased numbers of views. Recall that in the previous subsections, we propose a concept called a bundle of data streams $s = (s, \overline{s})$ which provides data to the two neural networks in the dual-view setting. Here, we will use multiple data stream bundles to provide data to different views so that the dual-view DCT can be adapted to the multi-view settings.

Specifically, we consider $n$ views $v_i(\cdot)$, $i = 1, .., n$ in the multi-view DCT. We assume that $n$ is an even number for the simplicity of presenting the multi-view algorithm. Next, we build $n/2$ independent data stream bundles $B = \big((s_1, \overline{s_1}), ..., (s_{n/2}, \overline{s_{n/2}})\big)$. Let $B_i(t)$ denote the training data that bundle $B_i$ provides at iteration $t$. Let $\mathcal{L}(v_i, v_j, B_k(t))$ denote the loss $\mathcal{L}$ in Step 6 of Algorithm 1 when dual training $v_i$ and $v_j$ using data $B_k(t)$. Then, at each iteration $t$, we consider

the training scheme implied by the following loss function

$$\mathcal{L}_{\text{fake } n\text{-view}}(t) = \sum_{i=1}^{n/2} \mathcal{L}(v_{2i-1}, v_{2i}, B_i(t)) \tag{3.7}$$

We call this *fake multi-view DCT* because Eq. 3.7 can be considered as $n/2$ independent dual-view DCTs. Next, we adapt Eq. 3.7 to the *real multi-view DCT*. In our multi-view DCT, at each iteration $t$, we consider an index list $l$ randomly shuffled from {1, 2, .., n}. Then, we use the following training loss function

$$\mathcal{L}_{n\text{-view}}(t) = \sum_{i=1}^{n/2} \mathcal{L}(v_{l_{2i-1}}, v_{l_{2i}}, B_i(t)) \tag{3.8}$$

Compared with Eq. 3.7, Eq. 3.8 randomly chooses a pair of views to train for each data stream bundle at each iteration. The benefits of this modeling are multifold. Firstly, Eq. 3.8 is converted from $n/2$ independent dual-view trainings; therefore, the hyperparameters for the dual-view setting are also suitable for multi-view settings. Thus, we can save our efforts in tuning parameters for a different number of views. Secondly, because of the relationship between Eq. 3.7 and Eq. 3.8, we can directly compare the training dynamics between different numbers of views. Thirdly, compared with computing the expected loss on all the possible pairs and data at each iteration, this modeling is also computationally efficient.

### 3.2.5 Implementation Details

To fairly compare with the previous state-of-the-art methods, we use the training and evaluation framework of Laine and Aila [144]. We port their implementation to PyTorch for easy multi-GPU support. Our multi-view implementation will automatically spread the models to different devices for maximal utilizations. For SVHN and CIFAR, we use a network architecture similar to [144]: we only change their weight normalization and mean-only batch normalization layers [233] to the natively supported batch normalization layers [124]. This change results in performances a little worse than but close to those reported in their paper. [144] thus is the most natural baseline. For ImageNet, we use a small model ResNet-18 [108] for fast experiments. In the

following, we introduce the datasets SVHN, CIFAR, and ImageNet, and how we train our models on them.

#### 3.2.5.1 SVHN

The Street View House Numbers (SVHN) dataset [197] contains real-world images of house numbers, each of which is of size $32 \times 32$. The label for each image is the centermost digit. Therefore, this is a classification problem with $10$ categories. Following Laine and Aila [144], we only use $1000$ images out of $73257$ official training images as the supervised part $\mathcal{S}$ to learn the models and the full test set of $26032$ images for testing. The rest $73257 - 1000$ images are considered as the unsupervised part $\mathcal{U}$. We train our method with the standard data augmentation, and our method significantly outperforms the previous state-of-the-art methods. Here, the data augmentation is only the random translation by at most $2$ pixels. We do not use any other types of data augmentations.

#### 3.2.5.2 CIFAR

CIFAR [141] has two image datasets, CIFAR-10 and CIFAR-100. Both of them contain color natural images of size $32 \times 32$, while CIFAR-10 includes $10$ categories and CIFAR-100 contains $100$ categories. Both of them have $50000$ images for training and $10000$ images for testing. Following Laine and Aila [144], for CIFAR-10, we only use $4000$ images out of $50000$ training images as the supervised part $\mathcal{S}$ and the rest $46000$ images are used as the unsupervised part $\mathcal{U}$. As for CIFAR-100, we use $10000$ images out of $50000$ training images as the supervised part $\mathcal{S}$ and the rest $40000$ images as the unsupervised part $\mathcal{U}$. We use the full $10000$ test images for evaluation for both CIFAR-10 and CIFAR-100. We train our methods with the standard data augmentation, which is the combination of random horizontal flip and translation by at most $2$ pixels.

#### 3.2.5.3 ImageNet

The ImageNet dataset contains about $1.3$ million natural color images for training and $50000$ images for validation. The dataset includes $1000$ categories, each of which typically has $1300$ images for training and $50$ for evaluation. Following the prior work that reported results on ImageNet [213], [231], [253], we uniformly choose $10\%$ data from $1.3$ million training images as supervised $\mathcal{S}$ and the rest as unsupervised $\mathcal{U}$. We report the single center crop error rates on the validation set. We train our models with data augmentation, which includes random resized crop to $224 \times 224$ and random horizontal flip. We do not use other advanced augmentation techniques such as color jittering or PCA lighting [142].

For SVHN and CIFAR, following [144], we use a warmup scheme for the hyperparameters $\lambda_{\text{cot}}$ and $\lambda_{\text{dif}}$. Specifically, we warm up them in the first $80$ epochs such that $\lambda = \lambda_{\text{max}} \cdot \exp(-5(1 - T/80)^2)$ when the epoch $T \leq 80$, and $\lambda_{\text{max}}$ after that. For SVHN and CIFAR, we set $\lambda_{\text{cot,max}} = 10$. For SVHN and CIFAR-10, $\lambda_{\text{dif,max}} = 0.5$, and for CIFAR-100 $\lambda_{\text{dif,max}} = 1.0$. For training, we train the networks using stochastic gradient descent with momentum $0.9$ and weight decay $0.0001$. The total number of training epochs is $600$ and we use a cosine learning rate schedule $lr = 0.05 \times (1.0 + \cos((T-1) \times \pi/600))$ at epoch $T$ [183]. The batch size is set to $100$ for SVHN, CIFAR-10 and CIFAR-100.

For ImageNet, we choose a different training scheme. Before using any data from $\mathcal{U}$, we first train two ResNet-18 individually with different initializations and training sequences on only the labeled data $\mathcal{S}$. Following ResNet [108], we train the models using stochastic gradient descent with momentum $0.9$, weight decay $0.0001$, and batch size $256$ for $600$ epochs, the time of which is the same as training $60$ epochs with full supervision. The learning rate is initialized as $0.1$ and multiplied by $0.1$ at the $301$st epoch. Then, we take the two pre-trained models to our unsupervised training loops. This time, we directly set $\lambda$ to the maximum values $\lambda = \lambda_{\text{max}}$ because the previous $600$ epochs have already warmed up the models. Here, $\lambda_{\text{cot,max}} = 1$ and $\lambda_{\text{dif,max}} = 0.1$. In the unsupervised loops, we use a cosine learning rate

| Method | SVHN | CIFAR-10 |
|---|---|---|
| GAN [232] | $8.11 \pm 1.30$ | $18.63 \pm 2.32$ |
| Stochastic Transformations [231] | – | $11.29 \pm 0.24$ |
| $\Pi$ Model [144] | $4.82 \pm 0.17$ | $12.36 \pm 0.31$ |
| Temporal Ensembling [144] | $4.42 \pm 0.16$ | $12.16 \pm 0.24$ |
| Mean Teacher [253] | $3.95 \pm 0.19$ | $12.31 \pm 0.28$ |
| Bad GAN [57] | $4.25 \pm 0.03$ | $14.41 \pm 0.30$ |
| VAT [192] | $3.86$ | $10.55$ |
| Deep Co-Training with 2 Views | $3.61 \pm 0.15$ | $9.03 \pm 0.18$ |
| Deep Co-Training with 4 Views | $3.38 \pm 0.05$ | $8.54 \pm 0.12$ |
| Deep Co-Training with 8 Views | $3.29 \pm 0.03$ | $8.35 \pm 0.06$ |

**Table 3-I.** Error rates on SVHN (1000 labeled) and CIFAR-10 (4000 labeled) benchmarks. Note that we report the averages of the single model error rates without ensembling them for the fairness of comparisons. We use architectures that are similar to that of $\Pi$ Model [144]. "–" means that the original papers did not report the corresponding error rates. We report means and standard deviations from 5 runs.

$lr = 0.005 \times (1.0 + \cos((T-1) \times \pi/20))$ and we train the networks for $20$ epochs on both $\mathcal{U}$ and $\mathcal{S}$. The batch size is set to $128$.

To make the loss $\mathcal{L}$ stable across different training iterations, we require that each data stream provides data batches whose proportions of the supervised data are close to the ratio of the size of $\mathcal{S}$ to the size of $\mathcal{D}$. To achieve this, we evenly divide the supervised and the unsupervised data to build each data batch in the data streams. As a result, the difference in the numbers of the supervised images between any two batches is no greater than $1$.

## 3.3 Results

In this section, we will present the experimental results on four datasets, *i.e.* SVHN [197], CIFAR-10, CIFAR-100 [141] and ImageNet [230]

### 3.3.1 SVHN and CIFAR-10

SVHN and CIFAR-10 are the datasets that the previous state-of-the-art methods for semi-supervised image recognition mostly focus on. Therefore, we first present the performances of

our method and show the comparisons with the previous state-of-the-art methods on these two datasets. Next, we will also provide ablation studies on the two datasets for better understandings of the dynamics and characteristics of dual-view and multi-view Deep Co-Training.

Table 3-I compares our method Deep Co-Training with the previous state-of-the-arts on SVHN and CIFAR-10 datasets. To make sure these methods are fairly compared, we do not ensemble the models of our method even though there are multiple well-trained models after the entire training procedure. Instead, we only report the average performances of those models. Compared with other state-of-the-art methods, Deep Co-Training achieves significant performance improvements when $2$, $4$, or $8$ views are used. As we will discuss in Section 4, all the methods listed in Table 3-I require implicit or explicit computations of multiple models, *e.g.* GAN [232] has a discriminative and a generative network, Bad GAN [57] adds another encoder network based on GAN, and Mean Teacher [253] has an additional EMA model. Therefore, the dual-view Deep Co-Training does not require more computations in terms of the total number of the networks.

Another trend we observe is that although $4$-view DCT gives significant improvements over $2$-view DCT, we do not see similar improvements when we increase the number of the views to $8$. For this observation, we speculate that this is because compared with $2$-views, $4$-views can use the majority vote rule when we encourage them to have close predictions on $\mathcal{U}$. When we increase the number of views to $8$, although it is expected to perform better, the advantages over $4$-views are not that strong compared with that of $4$-views over $2$-views. But $8$-view DCT converges faster than $4$-view DCT, which is even faster than dual-view DCT. The training dynamics of DCT with different numbers of views will be presented in the later subsections. We first provide our results on CIFAR-100 and ImageNet datasets in the next subsection.

| Method | CIFAR-100 | CIFAR-100+ |
|---|---|---|
| $\Pi$ Model [144] | $43.43 \pm 0.54$ | $39.19 \pm 0.36$ |
| Temporal Ensembling [144] | – | $38.65 \pm 0.51$ |
| Dual-View Deep Co-Training | $\mathbf{38.77 \pm 0.28}$ | $\mathbf{34.63 \pm 0.14}$ |

**Table 3-II.** Error rates on CIFAR-100 with $10000$ images labeled. Note that other methods listed in Table 3-I have not published results on CIFAR-100. The performances of our method are the averages of single model error rates of the networks without ensembling them for the fairness of comparisons. We use architectures that are similar to that of $\Pi$ Model [144]. "–" means that the original papers did not report the corresponding error rates. CIFAR-100+ and CIFAR-100 indicate that the models are trained with and without data augmentation, respectively. Our results are reported from $5$ runs.

| Method | Architecture | # Param | Top-1 | Top-5 |
|---|---|---|---|---|
| Stochastic Transformations [231] | AlexNet | 61.1M | – | 39.84 |
| VAE [213] with $10\%$ Supervised | Customized | 30.6M | 51.59 | 35.24 |
| Mean Teacher [253] | ResNet-18 | 11.6M | 49.07 | 23.59 |
| $100\%$ Supervised | ResNet-18 | 11.6M | 30.43 | 10.76 |
| $10\%$ Supervised | ResNet-18 | 11.6M | 52.23 | 27.54 |
| Dual-View Deep Co-Training | ResNet-18 | 11.6M | **46.50** | **22.73** |

**Table 3-III.** Error rates on the validation set of ImageNet benchmark with $10\%$ images labeled. The image size of our method in training and testing is $224 \times 224$.

### 3.3.2 CIFAR-100 and ImageNet

Compared with SVHN and CIFAR-10, CIFAR-100 and ImageNet are considered harder benchmarks [144] for the semi-supervised image recognition problem because their numbers of categories are 100 and 1000, respectively, greater than 10 categories in SVHN and CIFAR-10. Here, we provide our results on these two datasets. Table 3-II compares our method with the previous state-of-the-art methods that report the performances on CIFAR-100 dataset, *i.e.* $\Pi$ Model and Temporal Ensembling [144]. Dual-view Deep Co-Training even without data augmentation achieves similar performances with the previous state-of-the-arts that use data augmentation. When our method also uses data augmentation, the error rate drops significantly from $38.65$ to $34.63$. These results demonstrate the effectiveness of the proposed Deep Co-Training when the number of categories and the difficulty of the datasets increase.

Next, we show our results on ImageNet with $1000$ categories and $10\%$ labeled in Table 3-III. Our method has better performances than the supervised-only but is still behind the accuracy when $100\%$ supervision is used. When compared with the previous state-of-the-art methods, however, DCT shows significant improvements on both the Top-1 and Top-5 error rates. Here, the performances of [231] and [213] are quoted from their papers, and the performance of Mean Teacher [253] with ResNet-18 [108] is from running their official implementation on GitHub. When using the same architecture, DCT outperforms Mean Teacher by $\sim 2.6\%$ for Top-1 error rate, and $\sim 0.9\%$ for Top-5 error rate. Compared with [231] and [213] that use networks with more parameters and larger input size $256 \times 256$, Deep Co-Training also achieves lower error rates.

### 3.3.3 Ablation Study

In this subsection, we will provide several ablation studies for better understandings of our proposed Deep Co-Training method.

#### 3.3.3.1 On $\mathcal{L}_{\text{cot}}$ and $\mathcal{L}_{\text{dif}}$

Recall that the loss function used in Deep Co-Training has three parts, the supervision loss $\mathcal{L}_{\text{sup}}$, the co-training loss $\mathcal{L}_{\text{cot}}$ and the view difference constraint $\mathcal{L}_{\text{dif}}$. It is of interest to study the changes when the loss function $\mathcal{L}_{\text{cot}}$ and $\mathcal{L}_{\text{dif}}$ are used alone in addition to $\mathcal{L}_{\text{sup}}$ in $\mathcal{L}$. Fig. 3-1 shows the plots of the training dynamics of Deep Co-Training when different loss functions are used on SVHN and CIFAR-10 dataset. In both plots, the blue lines represent the loss function that we use in practice in training DCT, the green lines represent only the co-training loss $\mathcal{L}_{\text{cot}}$ and $\mathcal{L}_{\text{sup}}$ are applied, and the orange lines represent only the view difference constraint $\mathcal{L}_{\text{dif}}$ and $\mathcal{L}_{\text{sup}}$ are used. From Fig. 3-1, we can see that the Co-Training assumption ($\mathcal{L}_{\text{cot}}$) performs better at the beginning, but soon is overtaken by $\mathcal{L}_{\text{dif}}$. $\mathcal{L}_{\text{cot}}$ even falls into an extreme case in the SVHN dataset where its validation accuracy drops suddenly around the $400$-th epoch. For this phenomenon, we speculate that this is because the networks have collapsed into each

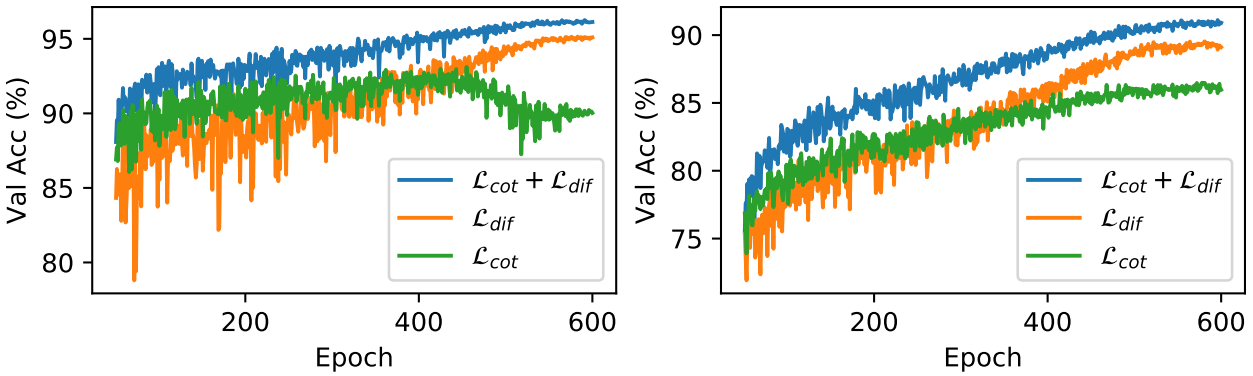



**Figure 3-1.** Ablation study on $\mathcal{L}_{\text{cot}}$ and $\mathcal{L}_{\text{dif}}$. The left plot is the training dynamics of dual-view Deep Co-Training on SVHN dataset, and the right is on CIFAR-10 dataset. "$\lambda_{\text{cot}}$", "$\lambda_{\text{dif}}$" represent the loss functions are used alone while "$\lambda_{\text{cot}} + \lambda_{\text{dif}}$" correspond to the weighted sum loss used in Deep Co-Training. In all the cases, $\mathcal{L}_{\text{sup}}$ is used.

other, which motivates us to investigate the dynamics of loss $\mathcal{L}_{\text{dif}}$. If our speculation is correct, there will also be abnormalities in loss $\mathcal{L}_{\text{dif}}$ around that epoch, which indeed we show in the next subsection. Moreover, this also supports our argument at the beginning of the chapter that a force to push models away is necessary for co-training multiple neural networks for semi-supervised learning. Another phenomenon we observe is that $\mathcal{L}_{\text{dif}}$ alone can achieve reasonable results. This is because when the adversarial algorithm fails to fool the networks, $\mathcal{L}_{\text{dif}}$ will degenrate to $\mathcal{L}_{\text{cot}}$. In other words, $\mathcal{L}_{\text{dif}}$ in practice combines the Co-Training assumption and View Difference Constraint, depending on the success rate of the adversarial algorithm.

#### 3.3.3.2 On the view difference

This is a sanity check on whether in dual-view training, two models tend to collapse into each other when we only model the Co-Training assumption, and if $\mathcal{L}_{\text{dif}}$ can push them away during training. To study this, we plot $\mathcal{L}_{\text{dif}}$ when it is minimized as in the Deep Co-Training and when it is not minimized, *i.e.* $\lambda_{\text{dif}} = 0$. Fig. 3-2 shows the plots of $\mathcal{L}_{\text{dif}}$ for SVHN dataset and CIFAR dataset, which correspond to the validation accuracies shown in Fig. 3-1. It is clear that when $\mathcal{L}_{\text{dif}}$ is not minimized as in the "$\mathcal{L}_{\text{cot}}$" case, $\mathcal{L}_{\text{dif}}$ is far greater than $0$, indicating that each model is vulnerable to the adversarial examples of the other. Like the extreme case we observe in Fig. 3-1 for SVHN dataset (left) around the $400$-th epoch, we also see a sudden increase of $\mathcal{L}_{\text{dif}}$

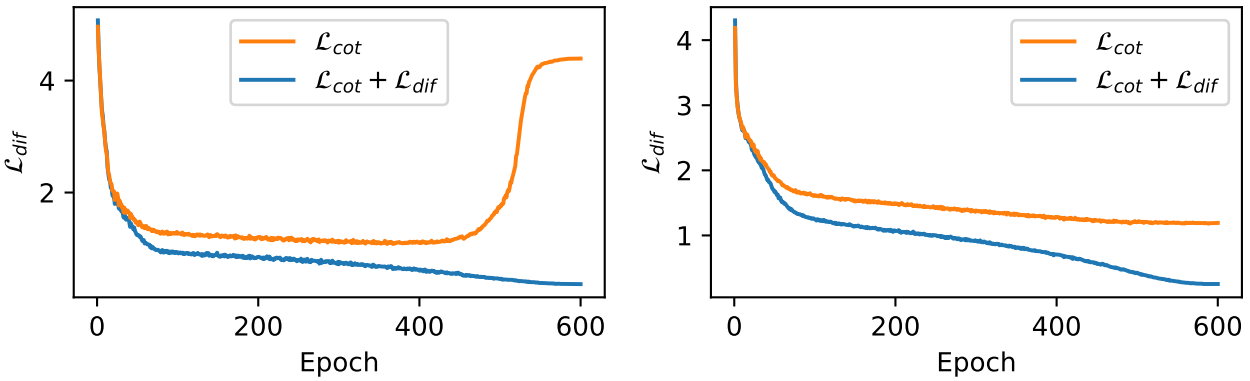



**Figure 3-2.** Ablation study on the view difference. The left plot is $\mathcal{L}_{\text{dif}}$ on SVHN dataset, and the right plot shows $\mathcal{L}_{\text{dif}}$ on CIFAR-10. Without minimizing $\mathcal{L}_{\text{dif}}$, $\mathcal{L}_{\text{dif}}$ is usually big in "$\mathcal{L}_{\text{cot}}$", indicating that the two models are making similar errors. In the SVHN dataset, the two models start to collapse into each other after around the $400$-th epoch because we observe a sudden increase of $\mathcal{L}_{\text{dif}}$. This corresponds to the sudden drop in the left plot of Fig. 3-1, which shows the relation between view difference and accuracy.

here in Fig. 3-2 for SVHN at similar epochs. This means that every adversarial example of one model fools the other model, *i.e.* they collapse into each other. The collapse directly causes a significant drop in the validation accuracy in the left of Fig. 3-1. These experimental results demonstrate the positive correlation between the view difference and the validation error. It also shows that the models in the dual-view training tend to collapse into each other when no force is applied to push them away. Finally, these results also support the effectiveness of our proposed $\mathcal{L}_{\text{dif}}$ as a loss function to increase the difference between the models.

#### 3.3.3.3 On the number of views

We have provided the performances of Deep Co-Training with different numbers of views for SVHN and CIFAR-10 datasets in Table 3-I, where we show that increasing the number of the views from 2 to 4 improves the performances of each individual model. But we also observe that the improvement becomes smaller when we further increase the number of views to 8. In Fig. 3-3, we show the training dynamics of Deep Co-Training when different numbers of views are trained simultaneously.

As shown in Fig. 3-3, we observe a faster convergence speed when we increase the number

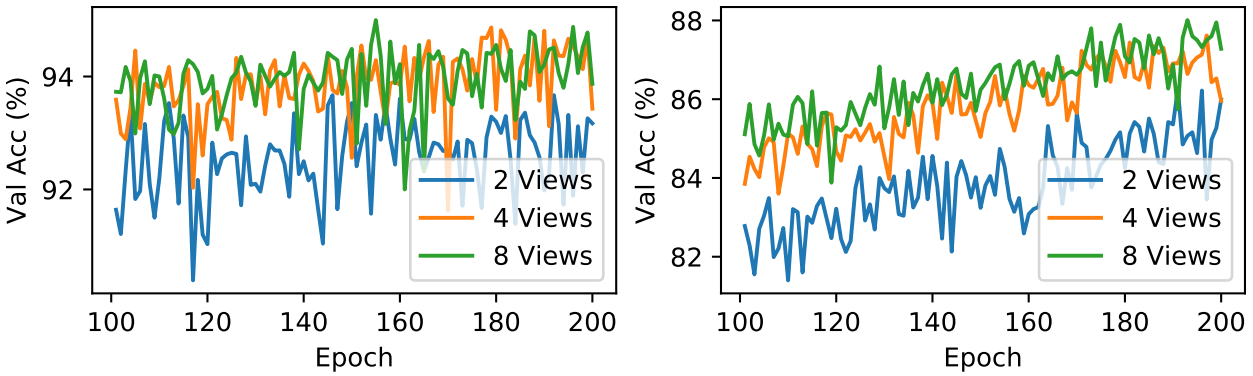



**Figure 3-3.** Training dynamics of Deep Co-Training with different numbers of views on SVHN dataset (left) and CIFAR-10 (right). The plots focus on the epochs from 100 to 200 where the differences are clearest. We observe a faster convergence speed when the number of views increases, but the improvements become smaller when the numbers of views increase from 4 to 8 compared with that from 2 to 4.

of views to train simultaneously. We focus on the epochs from 100 to 200 where the differences between different numbers of views are clearest. The performances of different views are directly comparable because of the scalability of the proposed multi-view Deep Co-Training. Like the improvements of $8$ views over $4$ views on the final validation accuracy, the improvements of the convergence speed also decrease compared with that of $4$ views over $2$ views.

## 3.4 Discussions

In this section, we discuss the relationship between Deep Co-Training and the previous methods. We also present perspectives alternative to the Co-Training framework for discussing Deep Co-Training.

### 3.4.1 Related Work

Deep Co-Training is also inspired by the recent advances in semi-supervised image recognition techniques [12], [144], [192], [222], [231] which train deep neural networks $f(\cdot)$ to be resistant to noises $\epsilon(z)$, *i.e.* $f(x) = f(x + \epsilon(z))$. We notice that their computations in one iteration require double feedforwardings and backpropagations, one for $f(x)$ and one for $f(x + \epsilon(z))$. We ask

the question: what would happen if we train two individual models as doing so requires the same amount of computations? We soon realized that training two models and encouraging them to have close predictions is related to the Co-Training framework [23], which has good theoretical results, provided that the two models are conditionally independent given the category. However, training models with only the Co-Training assumption is not sufficient for getting good performances because the models tend to collapse into each other, which is against the view difference between different models which is necessary for the Co-Training framework.

As stated in 2.2, we need a generative method to generate images on which two models predict differently. Generative Adversarial Networks (GANs) [6], [57], [232] are popular generative models for vision problems, and have also been used for semi-supervised image recognition. A problem of GANs is that they will introduce new networks to the Co-Training framework for generating images, which also need to be learned. Compared with GANs, Introspective Generative Models [126], [255] can generate images from discriminative models in a lightweight manner, which bears some similarities with the adversarial examples [94]. The generative methods that use discriminative models also include DeepDream [195], Neural Artistic Style [86], *etc*. We use adversarial examples in our Deep Co-Training for its natural applicability to avoiding models from collapsing into each other by training each model with the adversarial examples of the others.

Before the work discussed above, semi-supervised learning in general has already been widely studied. For example, the mutual-exclusivity loss used in [231] and the entropy minimization used in [192] resemble soft implementations of the self-training technique [79], [123], one of the earliest approaches for semi-supervised classification tasks. [328] provides a good survey for the semi-supervised learning methods in general.

### 3.4.2 Alternative Perspectives

In this subsection, we discuss the proposed Deep Co-Training method from several perspectives alternative to the Co-Training framework.

#### 3.4.2.1 Model Ensemble

Ensembling multiple independently trained models to get a more accurate and stable classifier is a widely used technique to achieve higher performances [26]. This is also applicable to deep neural networks [323], [325]. In other words, this suggests that when multiple networks with the same architecture are initialized differently and trained using data sequences in different time orders, they can achieve similar performances but in a complementary way [27]. In multi-view Deep Co-Training, we also train multiple models in parallel, but not independently, and our evaluation is done by taking one of them as the final classifier instead of averaging their predicted probabilities. Deep Co-Training in effect is searching for an initialization-free and data-order-free solution.

#### 3.4.2.2 Multi-Agent Learning

After the literature review of the most recent semi-supervised learning methods for image recognition, we find that almost all of them are within the multi-agent learning framework [203]. To name a few, GAN-based methods at least have a discriminative network and a generative network. Bad GAN [57] adds an encoder network based on GAN. The agents in GANs are interacting in an adversarial way. As we stated in Section 4.1, the methods that train deep networks to be resistant to noises also have the interacting behaviors as what two individual models would have, *i.e.* double feedforwardings and backpropagations. The agents in these methods are interacting in a cooperative way. Deep Co-Training explicitly models the cooperative multi-agent learning, which trains multiple agents from the supervised data and cooperative interactions between different agents. In the multi-agent learning framework, $\mathcal{L}_{\text{dif}}$ can be understood as learning from the errors of the others, and the loss function Eq. 3.8 resembles

the simulation of interactions within a crowd of agents.

#### 3.4.2.3 Knowledge Distillation

One characteristic of Deep Co-Training is that the models not only learn from the supervised data, but also learn from the predictions of the other models. This is reminiscent of knowledge distillation [112] where student models learn from teacher models instead of the supervisions from the datasets. In Deep Co-Training, all the models are students and learn from not only the predictions of the other student models but also the errors they make.

## 3.5 Conclusion

In this chapter, we present Deep Co-Training, a method for semi-supervised image recognition. It extends the Co-Training framework, which assumes that the data has two complementary views, based on which two effective classifiers can be built and are assumed to have close predictions on the unlabeled images. Motivated by the recent successes of deep neural networks in supervised image recognition, we extend the Co-Training framework to apply deep networks to the task of semi-supervised image recognition. In our experiments, we notice that the models are easy to collapse into each other, which violates the requirement of the view difference in the Co-Training framework. To prevent the models from collapsing, we use adversarial examples as the generative method to generate data on which the views have different predictions. The experiments show that this additional force that pushes models away is helpful for training and improves accuracies significantly compared with the Co-Training-only modeling.

Since Co-Training is a special case of multi-view learning, we also naturally extend the dual-view DCT to a scalable multi-view Deep Co-Training method where the hyperparameters for two views are also suitable for increased numbers of views. We test our proposed Deep Co-Training on the SVHN, CIFAR-10/100, and ImageNet datasets which are the benchmarks

that the previous state-of-the-art methods are tested on. Our method outperforms them by a large margin.

# Chapter 4

# Gradually Updated Neural Networks for Large-Scale Image Recognition

Depth is one of the keys that make neural networks succeed in the task of large-scale image recognition. The state-of-the-art network architectures usually increase the depths by cascading convolutional layers or building blocks. In this chapter, we present an alternative method to increase the depth. Our method is by introducing computation orderings to the channels within convolutional layers or blocks, based on which we gradually compute the outputs in a channel-wise manner. The added orderings not only increase the depths and the learning capacities of the networks without any additional computation costs but also eliminate the overlap singularities so that the networks are able to converge faster and perform better. Experiments show that the networks based on our method achieve state-of-the-art performances on CIFAR and ImageNet datasets.

## 4.1 Introduction

Deep neural networks have become the state-of-the-art systems for image recognition [108], [120], [142], [216], [239], [247], [269], [306] as well as other vision tasks [41], [90], [182], [217], [228], [238], [282]. The architectures keep going deeper, *e.g.*, from five convolutional layers [142] to $1001$ layers [109]. The benefit of deep architectures is their strong learning capacities because each new layer can potentially introduce more non-linearities and typically uses larger

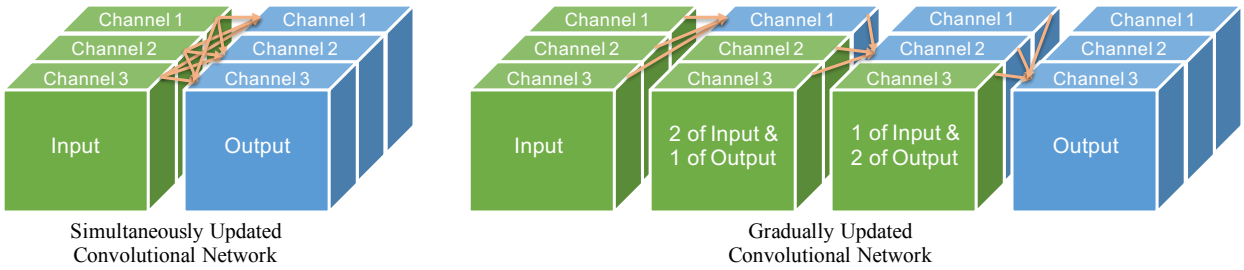



**Figure 4-1.** Comparing Simultaneously Updated Convolutional Network and Gradually Updated Convolutional Network. Left is a traditional convolutional network with three channels in both the input and the output. Right is our proposed convolutional network which decomposes the original computation into three sequential channel-wise convolutional operations. In our proposed GUNN-based architectures, the updates are done by *residual learning* [108], which we do not show in this figure.

receptive fields [239]. In addition, adding certain types of layers (e.g. [109]) will not harm the performance theoretically since they can just learn identity mapping. This makes stacking up layers more appealing in the network designs.

Although deeper architectures usually lead to stronger learning capacities, cascading convolutional layers (*e.g.* VGG [239]) or blocks (*e.g.* ResNet [108]) is not necessarily the only method to achieve this goal. In this chapter, we present a new way to increase the depth of the networks as an alternative to stacking up convolutional layers or blocks. Figure 4-1 provides an illustration that compares our proposed convolutional network that gradually updates the feature representations against the traditional convolutional network that computes its output simultaneously. By only adding an ordering to the channels without any additional computation, the later computed channels become deeper than the corresponding ones in the traditional convolutional network. We refer to the neural networks with the proposed computation orderings on the channels as Gradually Updated Neural Networks (GUNN). Figure 4-2 provides two examples of architecture designs based on cascading building blocks and GUNN. Without repeating the building blocks, GUNN increases the depths of the networks as well as their learning capacities.

It is clear that converting plain networks to GUNN increases the depths of the networks without any additional computations. What is less obvious is that GUNN in fact eliminates

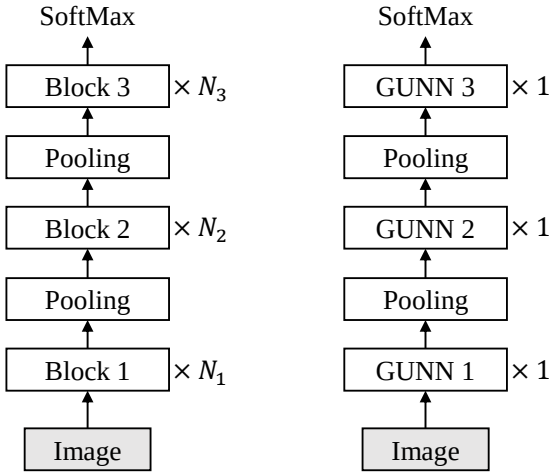



**Figure 4-2.** Comparing architecture designs based on cascading convolutional building blocks (left) and GUNN (right). Cascading-based architecture increases the depth by repeating the blocks. GUNN-based networks increase the depth by adding computation orderings to the channels of the building blocks.

the overlap singularities inherent in the loss landscapes of the cascading-based convolutional networks, which have been shown to adversely affect the training of deep neural networks as well as their performances [201], [272]. Overlap singularity is when internal neurons collapse into each other, *i.e.* they are unidentifiable by their activations. It happens in the networks, increases the training difficulties, and degrades the performances [201]. However, if a plain network is converted to GUNN, the added computation orderings will break the symmetry between the neurons. We prove that the internal neurons in GUNN are impossible to collapse into each other. As a result, the effective dimensionality can be kept during training and the model will be free from the degeneracy caused by collapsed neurons. Reflected in the training dynamics and the performances, this means that converting to GUNN will make the plain networks *easier to train* and *perform better*. Figure 4-3 compares the training dynamics of a 15-layer plain network on CIFAR-10 dataset [141] before and after converted to GUNN.

In this chapter, we test our proposed GUNN on highly competitive benchmark datasets, *i.e.* CIFAR [141] and ImageNet [230]. Experimental results demonstrate that our proposed GUNN-based networks achieve state-of-the-art performances compared with the previous

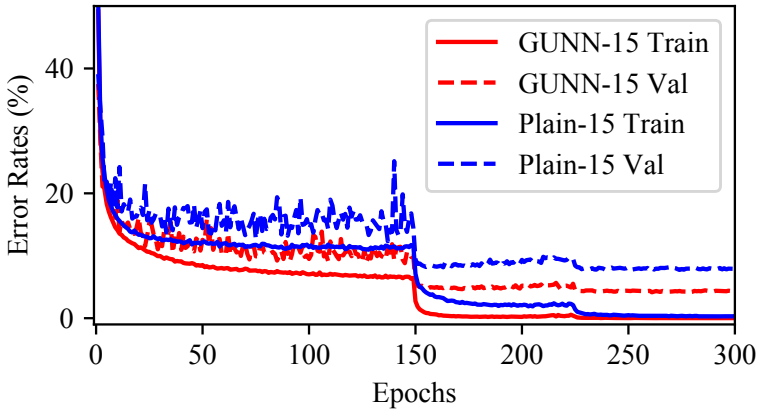



**Figure 4-3.** Training dynamics on CIFAR-10 dataset.

cascading-based architectures.

## 4.2 Related Work

The research focuses of image recognition have moved from feature designs [58], [184] to architecture designs [108], [120], [142], [236], [239], [247], [281], [306] due to the recent success of the deep neural networks. Highway Networks [246] proposed architectures that can be trained end-to-end with more than $100$ layers. The main idea of Highway Networks is to use bypassing paths. This idea was further investigated in ResNet [108], which simplifies the bypassing paths by using only identity mappings. As learning ultra-deep networks became possible, the depths of the models have increased tremendously. ResNet with pre-activation [109] and ResNet with stochastic depth [121] even managed to train neural networks with more than $1000$ layers. FractalNet [148] argued that in addition to summation, concatenation also helps train a deep architecture. More recently, ResNeXt [281] used group convolutions in ResNet and outperformed the original ResNet. DenseNet [120] proposed an architecture with dense connections by feature concatenation. Dual Path Net [49] finds a middle point between ResNet and DenseNet by concatenating them in two paths. Unlike the above cascading-based methods, GUNN eliminates the overlap singularities caused by the architecture symmetry. The detailed analyses can be found in Section 4.3.

Alternative to increasing the depth of the neural networks, another trend is to increase the widths of the networks. GoogleNet [247], [248] proposed an *Inception* module to concatenate feature maps produced by different filters. Following ResNet [108], the WideResNet [305] argued that compared with increasing the depth, increasing the width of the networks can be more effective in improving the performances. Besides varying the width and the depth, there are also other design strategies for deep neural networks [104], [140], [208], [222], [297]. Deeply-Supervised Nets [156] used auxiliary classifiers to provide direct supervision for the internal layers. Network in Network [170] adds micro perceptrons to the convolutional layers.

## 4.3 Model

### 4.3.1 Feature Update

We consider a feature transformation $\mathcal{F} : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$, where $n$ denotes the channel of the features and $m$ denotes the feature location on the $2$-D feature map. For example, $\mathcal{F}$ can be a convolutional layer with $n$ channels for both the input and the output. Let $\boldsymbol{x} \in \mathbb{R}^{m\times n}$ be the input and $\boldsymbol{y} \in \mathbb{R}^{m\times n}$ be the output, we have

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}) \tag{4.1}$$

Suppose that $\mathcal{F}$ can be decomposed into channel-wise transformation $\mathcal{F}_c(\cdot)$ that are independent with each other, then for any location $k$ and channel $c$ we have

$$y_c^k = \mathcal{F}_c(\boldsymbol{x}^{r(k)}) \tag{4.2}$$

where $\boldsymbol{x}^{r(k)}$ denotes the receptive field of the location $k$ and $\mathcal{F}_c$ denotes the transformation on channel $c$.

Let $U_C$ denote a feature update on channel set $C$, *i.e.*,

$$\begin{aligned} U_C(\boldsymbol{x}) :\ & y_c^k = \mathcal{F}_c(\boldsymbol{x}^{r(k)}), \forall c \in C, k \\ & y_c^k = x_c^k, \forall c \in \overline{C}, k \end{aligned} \tag{4.3}$$

Then, $U_C = \mathcal{F}$ when $C = \{1, ..., n\}$.

### 4.3.2 Gradually Updated Neural Networks

By defining the feature update $U_C$ on channel set $C$, the commonly used one-layer CNN is a special case of feature updates where every channel is updated simultaneously. However, we can also update the channels gradually. For example, the proposed GUNN can be formulated by

$$\begin{aligned} \text{GUNN}(\boldsymbol{x}) &= (U_{c_l} \circ U_{c_{(l-1)}} \circ ... \circ U_{c_2} \circ U_{c_1})(\boldsymbol{x}) \\ \text{where } \bigcup_{i=1}^{l} c_i &= \{1, 2, ..., n\} \text{ and } c_i \cap c_j = \Phi, \ \forall i \neq j \end{aligned} \tag{4.4}$$

When $l = 1$, GUNN is equivalent to $\mathcal{F}$.

Note that the number of parameters and computation of GUNN are the same as those of the corresponding $\mathcal{F}$ for any partitions $c_1, ..., c_l$ of $\{1, ..., n\}$. However, by decomposing $\mathcal{F}$ into channel-wise transformations and sequentially applying them, the later computed channels are deeper than the previous ones. As a result, the depth of the network can be increased, as well as the network's learning capacity.

### 4.3.3 Channel-wise Update by Residual Learning

**Algorithm 2:** Back-propagation for GUNN

**Input :** $U(\cdot) = (U_{c_l} \circ U_{c_{(l-1)}} \circ ... \circ U_{c_1})(\cdot)$, input $x$, output $y = U(x)$, gradients $\partial L/\partial y$, and parameters $\Theta$ for $U$.

**Output :** $\partial L/\partial \Theta$, $\partial L/\partial x$

$\partial L/\partial x \leftarrow \partial L/\partial y$
**for** $i \leftarrow l$ **to** $1$ **do**
  $y_c \leftarrow x_c, \forall c \in c_i$
  $\partial L/\partial y, \partial L/\partial \Theta_{c_i} \leftarrow \text{BP}(y, \partial L/\partial x, U_{c_i}, \Theta_{c_i})$
  $(\partial L/\partial x)_c \leftarrow (\partial L/\partial y)_c, \forall c \in c_i$
  $(\partial L/\partial x)_c \leftarrow (\partial L/\partial x)_c + (\partial L/\partial y)_c, \forall c \notin c_i$
**end**

We consider the residual learning proposed by ResNet [108] in our model. Specifically, we consider the channel-wise transformation $\mathcal{F}_c : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times 1}$ to be

$$\mathcal{F}_c(x) = \mathcal{G}_c(x) + x_c \tag{4.5}$$

where $\mathcal{G}_c$ is a convolutional neural network $\mathcal{G}_c : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times 1}$. The motivation of expressing $\mathcal{F}$ in a residual learning manner is to reduce overlap singularities [201], which will be discussed in Section 4.

### 4.3.4 Learning GUNN by Backpropagation

Here we show the backpropagation algorithm for learning the parameters in GUNN that uses the same amount of computations and memory as in $\mathcal{F}$. In Eq. 4.4, let the feature update $U_{c_i}$ be parameterized by $\Theta_{c_i}$. Let BP$(x, \partial L/\partial y, f, \Theta)$ be the back-propagation algorithm for differentiable function $y = f(x; \Theta)$ with the loss $L$ and the parameters $\Theta$. Algorithm 2 presents the back-propagation algorithm for GUNN. Since $U_{c_i}$ has the residual structures [108], the last two steps can be merged into

$$(\partial L/\partial x)_c \leftarrow (\partial L/\partial x)_c + (\partial L/\partial y)_c, \quad \forall c \tag{4.6}$$

which further simplifies the implementation. It is easy to see that converting networks to GUNN-based does not increase the memory usage in feed-forwarding. Given Algorithm 2, converting networks to GUNN will not affect the memory in both the training and the evaluation.

## 4.4 GUNN Eliminates Overlap Singularities

Overlap singularities are inherent in the loss landscapes of some network architectures which are caused by the non-identifiability of subsets of the neurons. They are identified and discussed in previous work [4], [201], [272], and are shown to be harmful to the performances of deep networks. Intuitively, overlap singularities exist in architectures where the internal neurons collapse into each other. As a result, the models are degenerate and the effective dimensionality is reduced. [201] demonstrated through experiments that residual learning (see Eq. 4.5) helps to reduce the overlap singularities in deep networks, which partly explains the exceptional performances of ResNet [108] compared with plain networks. In the following, we first use linear transformation as an example to demonstrate how GUNN-based networks break the overlap

singularities. Then, we generalize the results to ReLU DNN. Finally, we compare GUNN with the previous state-of-the-art network architectures from the perspective of singularity elimination.

### 4.4.1 Overlap Singularities in Linear Transformations

Consider a linear function $y = f(x) : \mathbb{R}^n \to \mathbb{R}^n$ such that

$$y_i = \sum_{j=1}^{n} \omega_{i,j} x_j, \quad \forall i \in \{1, .., n\} \tag{4.7}$$

Suppose that there exists a pair of collapsed neurons $y_p$ and $y_q$ ($p < q$). Then, for $\forall x$, $y_p = y_q$, and the equality holds after any number of gradient descents, *i.e.* $\Delta y_p = \Delta y_q$.

Eq. 4.7 describes a plain network. The solution for the existence of $y_p$ and $y_q$ is that $\omega_{p,j} = \omega_{q,j}, \forall j$. This is the case that is mostly discussed previously, which happens in the networks and degrades the performances.

When we add the residual learning, Eq. 4.7 becomes

$$y_i = x_i + \sum_{j=1}^{n} \omega_{i,j} x_j, \quad \forall i \in \{1, .., n\} \tag{4.8}$$

Collapsed neurons require that $\omega_{p,p} + 1 = \omega_{q,p}$, $\omega_{q,q} + 1 = \omega_{p,q}$. This will make the collapse of $y_p$ and $y_q$ very hard when $\omega$ is initialized from a normal distribution $\mathcal{N}(0, \sqrt{2/n})$ as in ResNet, but still possible.

Next, we convert Eq. 4.8 to GUNN, *i.e.,*

$$y_i = x_i + \sum_{j=1}^{i-1} \omega_{i,j} y_j + \sum_{j=i}^{n} \omega_{i,j} x_j, \quad \forall i \in \{1, .., n\} \tag{4.9}$$

Suppose that $y_p$ and $y_q$ ($p < q$) collapse. Consider $\Delta y$, the value difference at $x$ after one step of gradient descent on $\omega$ with input $x$, $\partial L / \partial y$ and learning rate $\epsilon$. When $\epsilon \to 0$,

$$\Delta y_i = \epsilon \frac{\partial L}{\partial y_i} (\sum_{j=1}^{i-1} y_j^2 + \sum_{j=i}^{n} x_j^2) + \sum_{j=1}^{i-1} \omega_{i,j} \Delta y_j \tag{4.10}$$

As $\Delta y_p = \Delta y_q, \forall x$, we have $\omega_{q,j} = 0, \quad \forall j : p < j < q$. But this condition will be broken in the next update; thus, $q = p + 1$. Then, we derive that $y_p = y_q = 0$. But these will also be broken in the next step of gradient descent optimization. Hence, $y_p$ and $y_q$ cannot collapse into each other.

### 4.4.2 Overlap Singularities in ReLU DNN

In practice, architectures are usually composed of several linear layers and non-linearity layers. Analyzing all the possible architectures is beyond our scope. Here, we discuss the commonly used ReLU DNN, in which only linear transformations and ReLUs are used by simple layer cascading.

Following the notations in §4.3, we use $y = \mathcal{G}(x) + x$, in which $\mathcal{G}(x)$ is a ReLU DNN. Note that $\mathcal{G}$ is a continuous piecewise linear (PWL) function [7], which means that there exists a finite set of polyhedra whose union is $\mathbb{R}^n$, and $\mathcal{G}$ is affine linear over each polyhedron.

Suppose that we convert $\mathcal{G}(x) + x$ to GUNN and there exists a pair of collapsed neurons $y_p$ and $y_q$ ($p < q$). Then, the set of polyhedra for $y_p$ is the same as for $y_q$. Let $\mathbb{P}$ be a polyhedron for $y_p$ and $y_q$ defined above. Then, $\forall x, \mathbb{P}, i$,

$$y_i = x_i + \sum_{j=1}^{i-1} \omega_{i,j}(\mathbb{P}) y_j + \sum_{j=i}^{n} \omega_{i,j}(\mathbb{P}) x_j \tag{4.11}$$

where $\omega(\mathbb{P})$ denotes the parameters for polyhedron $\mathbb{P}$. Note that on each $\mathbb{P}$, $y$ is a function of $x$ in the form of Eq. 4.9; hence, $y_p$ and $y_q$ cannot collapse into each other. Since the union of all polyhedra is $\mathbb{R}^n$, we conclude that GUNN eliminates the overlap singularities in ReLU DNN.

### 4.4.3 Discussions and Comparisons

The previous two subsections consider the GUNN conversion where $|c_i| = 1, \forall i$ (see Eq. 4.4). But this will slow down the computation on GPU due to the data dependency. Without specialized hardware or library support, we decide to increase $|c_i|$ to $> 10$. The resulting models run at the speed between ResNeXt [281] and DenseNet [120]. But this change introduces singularities into the channels from the same set $c_i$. Then, the residual learning helps GUNN to reduce the singularities within the same set $c_i$ since we initialize the parameters from a normal distribution $\mathcal{N}(0, \sqrt{2/n})$. We will compare the results of GUNN with and without residual learning in the experiments.

We compare GUNN with the state-of-the-art architectures from the perspective of overlap singularities. ResNet [108] and its variants use residual learning, which reduces but cannot eliminate the singularities. ResNeXt [281] uses group convolutions to break the symmetry between groups, which further helps to avoid neuron collapses. DenseNet [120] concatenates the outputs of layers as the input to the next layer. DenseNet and GUNN both create dense connections, while DenseNet reuses the outputs by concatenating and GUNN by adding them back to the inputs. But the channels within the same layer of DenseNet are still possible to collapse into each other since they are symmetric. In contrast, adding back makes residual learning possible in GUNN. This makes residual learning indispensable in GUNN-based networks.

## 4.5 Network Architectures

In this section, we will present the details of our architectures for the CIFAR [141] and ImageNet [230] datasets.

### 4.5.1 Simultaneously Updated Neural Networks and Gradually Updated Neural Networks

Since the proposed GUNN is a method for increasing the depths of the convolutional networks, specifying the architectures to be converted is equivalent to specifying the GUNN-based architectures. The architectures before conversion, the Simultaneously Updated Neural Networks (SUNN), become natural baselines for our proposed GUNN networks. We first study what baseline architectures can be converted.

There are two assumptions about the feature transformation $\mathcal{F}$ (see Eq. 4.1): (1) the input and the output sizes are the same, and (2) $\mathcal{F}$ is channel-wise decomposable. To satisfy the first assumption, we will first use a convolutional layer with Batch Normalization [124] and ReLU [196] to transform the feature space to a new space where the number of the channels is wanted. To satisfy the second assumption, instead of directly specifying the transform $\mathcal{F}$, we focus on designing $\mathcal{F}_{c_i}$, where $c_i$ is a subset of the channels (see Eq. 4.4). To be consistent with

the term *update* used in GUNN and SUNN, we refer to $\mathcal{F}_{c_i}$ as the *update units* for channels $c_i$.

#### 4.5.1.1 Bottleneck Update Units

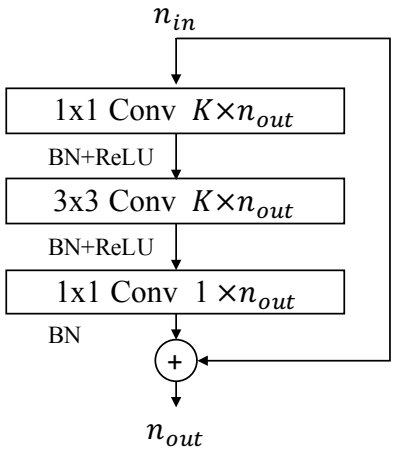



**Figure 4-4.** Bottleneck Update Units for both SUNN and GUNN.

In the architectures proposed in this chapter, we adopt bottleneck neural networks as shown in Figure 4-4 for the update units for both the SUNN and GUNN. Suppose that the update unit maps the input features of channel size $n_{\text{in}}$ to the output features of size $n_{\text{out}}$. Each unit contains three convolutional layers. The first convolutional layer transforms the input features to $K \times n_{\text{out}}$ using a $1 \times 1$ convolutional layer. The second convolutional layer is of kernel size $3 \times 3$, stride $1$, and padding $1$, outputting the features of size $K \times n_{\text{out}}$. The third layer computes the features of size $n_{\text{out}}$ using a $1 \times 1$ convolutional layer. The output is then added back to the input, following the residual architecture proposed in ResNet [108]. We add batch normalization layer [124] and ReLU layer [196] after the first and the second convolutional layers, while only adding batch normalization layer after the third layer. Stacking up $M$ update units also generates a new one. In total, we have two hyperparameters for designing an update unit: the expansion rate $K$ and the number of the $3$-layer update units $M$.

| Stage | Output | WideResNet-28-10 | GUNN-15 |
|---|---|---|---|
| Conv1 | $32 \times 32$ | $\begin{bmatrix} 3 \times 3, 16 \end{bmatrix}$ | $\begin{bmatrix} 3 \times 3, 64 \\ 1 \times 1, 240 \end{bmatrix}$ |
| Conv2* | $32 \times 32$ | $\begin{bmatrix} 3 \times 3, 160 \\ 3 \times 3, 160 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans1 | $16 \times 16$ | – | $\begin{bmatrix} 1 \times 1, 300 \\ \text{AvgPool} \end{bmatrix}$ |
| Conv3* | $16 \times 16$ | $\begin{bmatrix} 3 \times 3, 320 \\ 3 \times 3, 320 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans2 | $8 \times 8$ | – | $\begin{bmatrix} 1 \times 1, 360 \\ \text{AvgPool} \end{bmatrix}$ |
| Conv4* | $8 \times 8$ | $\begin{bmatrix} 3 \times 3, 640 \\ 3 \times 3, 640 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans3 | $1 \times 1$ | $\begin{bmatrix} \text{GAPool} \\ \text{fc, softmax} \end{bmatrix}$ | $\begin{bmatrix} 1 \times 1, 360 \\ \text{GAPool} \\ \text{fc, softmax} \end{bmatrix}$ |
| GPU Memory | | 4.903GB@64 | 2.485GB@64 |
| # Params | | 36.5M | 1.6M |
| Error (C10/C100) | | 4.17 / 20.50 | 4.15 / 20.45 |

**Table 4-I.** Architecture comparison between WideResNet-28-10 [305] and GUNN-15 for CIFAR. (Left) WideResNet-28-10. (Right) GUNN-15. GUNN achieves comparable accuracies on CIFAR10/100 while using a smaller number of parameters and consuming less GPU memory during training. In GUNN-15, the convolution stages with stars are computed using GUNN while others are not.

#### 4.5.1.2 One Resolution, One Representation

Our architectures will have only one representation at one resolution besides the pooling layers and the convolutional layers that initialize the needed numbers of channels. Take the architecture in Table 4-I as an example. There are two processes for each resolution. The first one is the transition process, which computes the initial features with the dimensions of the next resolution, then downsamples it to $1/4$ using a $2 \times 2$ average pooling. A convolutional operation is needed here because $\mathcal{F}$ is assumed to have the same input and output sizes. The

next process is using GUNN to update this feature space gradually. Each channel will only be updated once, and all channels will be updated after this process. Unlike most of the previous networks, after these two processes, the feature transformations at this resolution are complete. There will be no more convolutional layers or blocks following this feature representation, *i.e.*, *one resolution, one representation*. Then, the network will compute the initial features for the next resolution, or compute the final vector representation of the entire image by a global average pooling. By designing networks in this way, SUNN networks usually have about $20$ layers before converting to GUNN-based networks.

#### 4.5.1.3 Channel Partitions

With the clearly defined update units, we can easily build SUNN and GUNN layers by using the units to update the representations following Eq. 4.4. The hyperparameters for the SUNN/GUNN layer are the number of the channels $N$ and the partition over those channels. In our proposed architectures, we evenly partition the channels into $P$ segments. Then, we can use $N$ and $P$ to represent the configuration of a layer. Together with the hyperparameters in the update units, we have four hyperparameters to tune for one SUNN/GUNN layer, *i.e.* $\{N, P, K, M\}$.

### 4.5.2 Architectures for CIFAR

We have implemented two neural networks based on GUNN to compete with the previous state-of-the-art methods on CIFAR datasets, *i.e.*, GUNN-$15$ and GUNN-$24$. Table 4-I shows the big picture of GUNN-$15$. Here, we present the details of the hyperparameter settings for GUNN-$15$ and GUNN-$24$. For GUNN-$15$, we have three GUNN layers, Conv$2$, Conv$3$ and Conv$4$. The configuration for Conv$2$ is $\{N = 240, P = 20, K = 2, M = 1\}$, the configuration for Conv$3$ is $\{N = 300, P = 25, K = 2, M = 1\}$, and the configuration for Conv$4$ is $\{N = 360, P = 30, K = 2, M = 1\}$. For GUNN-$24$, based on GUNN-$15$, we change the number of output channels of Conv$1$ to $720$, Trans$1$ to $900$, Trans$2$ to $1080$, and Trans$3$ to $1080$. The hyperparameters are $\{N = 720, P = 20, K = 3, M = 2\}$ for Conv$2$, $\{N = 900, P = 25, K = 3, M = 2\}$ for Conv$3$,

| Stage | Output | ResNet-152 | GUNN-18 |
|---|---|---|---|
| Conv1 | $56 \times 56$ | $\begin{bmatrix} 7 \times 7, 64, 2 \\ 3 \times 3 \text{ MaxPool}, 2 \end{bmatrix}$ | $\begin{bmatrix} 7 \times 7, 64, 2 \\ 3 \times 3 \text{ MaxPool}, 2 \\ 1 \times 1, 400 \end{bmatrix}$ |
| Conv2* | $56 \times 56$ | $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans1 | $28 \times 28$ | – | $\begin{bmatrix} 1 \times 1, 800 \\ \text{AvgPool} \end{bmatrix}$ |
| Conv3* | $28 \times 28$ | $\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans2 | $14 \times 14$ | – | $\begin{bmatrix} 1 \times 1, 1600 \\ \text{AvgPool} \end{bmatrix}$ |
| Conv4* | $14 \times 14$ | $\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans3 | $7 \times 7$ | – | $\begin{bmatrix} 1 \times 1, 2000 \\ \text{AvgPool} \end{bmatrix}$ |
| Conv5* | $7 \times 7$ | $\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$ |
| Trans4 | $1 \times 1$ | $\begin{bmatrix} \text{GAPool} \\ \text{fc, softmax} \end{bmatrix}$ | $\begin{bmatrix} \text{GAPool} \\ \text{fc, softmax} \end{bmatrix}$ |
| # Params | | 60.2M | 28.9M |
| Error (Top-1/5) | | 22.2 / 6.2 | 21.65 / 5.87 |

**Table 4-II.** Architecture comparison between ResNet [108] and GUNN-18 for ImageNet-152. (Left) ResNet-152. (Right) GUNN-18. GUNN achieves better accuracies on ImageNet while using a smaller number of parameters.

and $\{N = 1080, P = 30, K = 3, M = 2\}$ for Conv3. The number of parameters of GUNN-15 is $1585746$ for CIFAR-10 and $1618236$ for CIFAR-100. The number of parameters of GUNN-24 is $29534106$ for CIFAR-10 and $29631396$ for CIFAR-100. The GUNN-15 is aimed to compete with the methods published in an early stage by using a much smaller model, while GUNN-24 is targeted at comparing with ResNeXt [281] and DenseNet [120] to get the state-of-the-art performance.

### 4.5.3 Architectures for ImageNet

We implement a neural network GUNN-$18$ to compete with the state-of-the-art neural networks on ImageNet with a similar number of parameters. Table 4-II shows the big picture of the neural network architecture of GUNN-$18$. Here, we present the detailed hyperparameters for the GUNN layers in GUNN-$18$. The GUNN layers include Conv2, Conv3, Conv4 and Conv5. The hyperparameters are $\{N = 400, P = 10, K = 2, M = 1\}$ for Conv2, $\{N = 800, P = 20, K = 2, M = 1\}$ for Conv3, $\{N = 1600, P = 40, K = 2, M = 1\}$ for Conv4 and $\{N = 2000, P = 50, K = 2, M = 1\}$ for Conv5. The number of parameters is $28909736$. The GUNN-$18$ is targeted at competing with the previous state-of-the-art methods that have similar numbers of parameters, *e.g.*, ResNet-$50$ [281], ResNeXt-$50$ [281] and DenseNet-$264$ [120].

We also implement a wider GUNN-based neural networks *Wide-GUNN-18* for better capacities. The hyperparameters are $\{N = 1200, P = 30, K = 2, M = 1\}$ for Conv2, $\{N = 1600, P = 40, K = 2, M = 1\}$ for Conv3, $\{N = 2000, P = 50, K = 2, M = 1\}$ for Conv4 and $\{N = 2000, P = 50, K = 2, M = 1\}$ for Conv5. The number of parameters is $45624936$. The Wide-GUNN-$18$ is targeted at competing with ResNet-$101$, ResNext-$101$, DPN [49] and SENet [118].

## 4.6 Experiments

In this section, we demonstrate the effectiveness of the proposed GUNN on several benchmark datasets.

### 4.6.1 Benchmark Datasets

#### 4.6.1.1 CIFAR

CIFAR [141] has two color image datasets: CIFAR-10 (C10) and CIFAR-100 (C100). Both datasets consist of natural images with the size of $32 \times 32$ pixels. The CIFAR-10 dataset has $10$ categories, while the CIFAR-100 dataset has $100$ categories. For both of the datasets, the

training and test set contain $50,000$ and $10,000$ images, respectively. To fairly compare our method with the state-of-the-arts [108], [120], [121], [148], [156], [170], [229], [245], [246], [281], we use the same training and testing strategies, as well as the data processing methods. Specifically, we adopt a commonly used data augmentation scheme, *i.e.,* mirroring and shifting, for these two datasets. We use channel means and standard derivations to normalize the images for data pre-processing.

#### 4.6.1.2 ImageNet

The ImageNet dataset [230] contains about $1.28$ million color images for training and $50,000$ for validation. The dataset has $1000$ categories. We adopt the same data augmentation methods as in the state-of-the-art architectures [108], [109], [120], [281] for training. For testing, we use single crop at the size of $224 \times 224$. Following the state-of-the-arts [108], [109], [120], [281], we report the validation error rates.

### 4.6.2 Training Details

We train all of our networks using stochastic gradient descents. On CIFAR-10/100 [141], the initial learning rate is set to $0.1$, the weight decay is set to $1e^{-4}$, and the momentum is set to $0.9$ without dampening. We train the models for $300$ epochs. The learning rate is divided by $10$ at $150$th epoch and $225$th epoch. We set the batch size to $64$, following [120]. All the results reported for CIFAR, regardless of the detailed configurations, were trained using $4$ NVIDIA Titan X GPUs with data parallelism. On ImageNet [230], the learning rate is also set to $0.1$ initially, and decreases following the schedule in DenseNet [120]. The batch size is set to $256$. The network parameters are also initialized following [108]. We use $8$ Tesla V100 GPUs with the data parallelism to get the reported results. Our results are directly comparable with ResNet, WideResNet, ResNeXt and DenseNet.

| Method | | | C10 | C100 |
|---|---|---|---|---|
| Network in Network [170] | | | 8.81 | – |
| All-CNN [245] | | | 7.25 | 33.71 |
| Deeply Supervised Network [156] | | | 7.97 | 34.57 |
| Highway Network [246] | | | 7.72 | 32.39 |
| | # layers | # params | | |
| ResNet [108], [121] | 110 | 1.7M | 6.41 | 27.22 |
| FractalNet [148] | 21 | 38.6M | 5.22 | 23.30 |
| Stochastic Depth [121] | 1202 | 10.2M | 4.91 | 24.58 |
| ResNet with pre-act [109] | 1001 | 10.2M | 4.62 | 22.71 |
| WideResNet-28-10 [305] | 28 | 36.5M | 4.17 | 20.50 |
| WideResNet-40-10 [305] | 40 | 55.8M | 3.80 | 18.30 |
| ResNeXt [281] | 29 | 68.1M | 3.58 | 17.31 |
| DenseNet [120] | 190 | 25.6M | 3.46 | 17.18 |
| Snapshot Ensemble [119] | – | 163.2M | 3.44 | 17.41 |
| GUNN-15 | 15 | 1.6M | 4.15 | 20.45 |
| GUNN-24 | 24 | 29.6M | **3.21** | **16.69** |
| GUNN-24 Ensemble | $24 \times 6$ | 177.6M | **3.02** | **15.61** |

**Table 4-III.** Classification errors (%) on the CIFAR-10/100 test set. All methods are with data augmentation. The third group shows the most recent state-of-the-art methods. The performances of GUNN are presented in the fourth group. A very small model GUNN-15 outperforms all the methods in the second group except WideResNet-40-10. A relatively bigger model GUNN-24 surpasses all the competing methods. GUNN-24 becomes more powerful with ensemble [119].

| Method | # layers | # params | C10 | C100 |
|---|---|---|---|---|
| GUNN-15-NoRes | 15 | 1.6M | 4.45 | 21.15 |
| GUNN-15 | 15 | 1.6M | 4.15 | 20.45 |
| SUNN-15 | 15 | 1.6M | 5.64 | 23.75 |
| GUNN-15 | 15 | 1.6M | 4.15 | 20.45 |
| SUNN-24 | 24 | 29.6M | 3.88 | 19.60 |
| GUNN-24 | 24 | 29.6M | 3.21 | 16.69 |

**Table 4-IV.** Ablation study on residual learning and SUNN.

### 4.6.3 Results on CIFAR

We train two models GUNN-$15$ and GUNN-$24$ for the CIFAR-10/100 dataset. Table 4-III shows the comparisons between our method and the previous state-of-the-art methods. Our method GUNN achieves the best results in the test of both the single model and the ensemble test. Here, we use Snapshot Ensemble [119].

#### 4.6.3.1 Baseline Methods

Here we present the details of baseline methods in Table 4-III. The performances of ResNet [108] are reported in Stochastic Depth [121] for both C10 and C100. The WideResNet [305] WRN-$40$-$10$ is reported in their official code repository on GitHub. The ResNeXt in the third group is of configuration $16 \times 64$d, which has the best result reported in the paper [281]. The DenseNet is of configuration DenseNet-BC ($k = 40$), which achieves the best performances on CIFAR-10/100. The Snapshot Ensemble [119] uses $6$ DenseNet-$100$ to ensemble during inference. We do not compare with methods that use more data augmentation (*e.g.* [308]) or stronger regularizations (*e.g.* [85]) for the fairness of comparison.

#### 4.6.3.2 Ablation Study

For the ablation study, we compare GUNN with SUNN, *i.e.*, the networks before the conversion. Table 4-IV shows the comparison results, which demonstrate the effectiveness of GUNN. We also compare the performances of GUNN with and without residual learning.

| Method | # layers | # params | top-1 | top-5 |
|---|---|---|---|---|
| VGG-16 [239] | 16 | 138M | 28.5 | 9.9 |
| ResNet-50 [108] | 50 | 25.6M | 24.0 | 7.0 |
| ResNeXt-50 [281] | 50 | 25.0M | 22.2 | 6.0 |
| DenseNet-264 [120] | 264 | 33.3M | 22.15 | 6.12 |
| SUNN-18* | 18 | 28.9M | 26.16 | 8.48 |
| GUNN-18* | 18 | 28.9M | **21.65** | **5.87** |
| ResNet-101 [108] | 101 | 44.5M | 22.0 | 6.0 |
| ResNeXt-101 [281] | 101 | 44.1M | 21.2 | 5.6 |
| DPN-98 [49] | 98 | 37.7M | 20.73 | 5.37 |
| SE-ResNeXt-101 [118] | 101 | 49.0M | 20.70 | **5.01** |
| Wide GUNN-18* | 18 | 45.6M | **20.59** | 5.52 |

**Table 4-V.** Single-crop classification errors (%) on the ImageNet validation set. The test size of all the methods is $224 \times 224$. Ours: $*$.

#### 4.6.3.3 Results on ImageNet

We evaluate the GUNN on the ImageNet classification task, and compare our performances with the state-of-the-art methods. These methods include VGGNet [239], ResNet [108], ResNeXt [281], DenseNet [120], DPN [49] and SENet [118]. The comparisons are shown in Table 4-V. The results of ours, ResNeXt, and DenseNet are directly comparable as these methods use the same framework for training and testing networks. Table 4-V groups the methods by their numbers of parameters, except VGGNet which has $1.38 \times 10^8$ parameters.

The results presented in Table 4-V demonstrate that with similar numbers of parameters, GUNN can achieve comparable performances with the previous state-of-the-art methods. For GUNN-18, we also conduct an ablation experiment by comparing the corresponding SUNN with GUNN of the same configuration. Consistent with the experimental results on the CIFAR-10/100 dataset, the proposed GUNN improves the accuracy on ImageNet dataset.

## 4.7 Conclusion

In this chapter, we propose Gradually Updated Neural Network (GUNN), a novel, simple yet effective method to increase the depths of neural networks as an alternative to cascading layers.

GUNN is based on Convolutional Neural Networks (CNNs), but differs from CNNs in the way of computing outputs. The outputs of GUNN are computed gradually rather than simultaneously as in CNNs in order to increase the depth. Essentially, GUNN assumes the input and the output are of the same size and adds a computation ordering to the channels. The added ordering increases the receptive fields and non-linearities of the later computed channels. Moreover, it eliminates the overlap singularities inherent in the traditional convolutional networks. We test GUNN on the task of image recognition. The evaluations are done in three highly competitive benchmarks, CIFAR-10, CIFAR-100, and ImageNet. The experimental results demonstrate the effectiveness of the proposed GUNN on image recognition. In the future, since the proposed GUNN can be used to replace CNNs in other neural networks, we will study the applications of GUNN in other visual tasks, such as object detection and semantic segmentation.

# Chapter 5

# Neural Rejuvenation: Improving Deep Network Training by Enhancing Computational Resource Utilization

In this chapter, we study the problem of improving computational resource utilization of neural networks. Deep neural networks are usually over-parameterized for their tasks in order to achieve good performances, thus are likely to have underutilized computational resources. This observation motivates a lot of research topics, *e.g.*, network pruning, architecture search, *etc*. As models with higher computational costs (*e.g.* more parameters or more computations) usually have better performances, we study the problem of improving the resource utilization of neural networks so that their potentials can be further realized. To this end, we propose a novel optimization method named Neural Rejuvenation. As its name suggests, our method detects dead neurons and computes resource utilization in real time, rejuvenates dead neurons by resource reallocation and reinitialization, and trains them with new training schemes. By simply replacing standard optimizers with Neural Rejuvenation, we are able to improve the performances of neural networks by a very large margin while using similar training efforts and maintaining their original resource usages.

## 5.1 Introduction

Deep networks achieve state-of-the-art performances in many visual tasks [41], [108], [190]. On large-scale tasks such as ImageNet [230] classification, a common observation is that the models with more parameters, or more FLOPs, tend to achieve better results. For example, DenseNet [120] plots the validation error rates as functions of the number of parameters and FLOPs, and shows consistent accuracy improvements as the model size increases. This is consistent with our intuition that large-scale tasks require models with sufficient capacity to fit the data well. As a result, it is usually beneficial to train a larger model if the additional computational resources are properly utilized. However, previous work on network pruning [181], [300] already shows that many neural networks trained by SGD have unsatisfactory resource utilization. For instance, the number of parameters of a VGG [239] network trained on CIFAR [141] can be compressed by a factor of 10 without affecting its accuracy [181]. Such low utilization results in a waste of training and testing time, and restricts the models from achieving their full potentials. To address this problem, we investigate novel neural network training and optimization techniques to enhance resource utilization and improve accuracy.

Formally, this chapter studies the following optimization problem. We are given a loss function $\mathcal{L}(f(x;\mathcal{A},\theta_{\mathcal{A}}),y)$ defined on data $(x,y)$ from a dataset $\mathcal{D}$, and a computational resource constraint $\mathcal{C}$. Here, $f(x;\mathcal{A},\theta_{\mathcal{A}})$ is a neural network with architecture $\mathcal{A}$ and parameterized by $\theta_{\mathcal{A}}$. Let $c(\mathcal{A})$ denote the cost of using architecture $\mathcal{A}$ in $f$, *e.g.*, $c(\mathcal{A})$ can be the number of parameters in $\mathcal{A}$ or its FLOPs. Our task is to find $\mathcal{A}$ and its parameter $\theta_{\mathcal{A}}$ that minimize the average loss $\mathcal{L}$ on dataset $\mathcal{D}$ under the resource constraint $\mathcal{C}$, *i.e.*,

$$\begin{aligned} \mathcal{A},\theta_{\mathcal{A}} = \arg\min_{\mathcal{A},\theta_{\mathcal{A}}} \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\big(f(x_i;\mathcal{A},\theta_{\mathcal{A}}),y_i\big) \\ \textit{s.t.}\ \ c(\mathcal{A}) \le \mathcal{C} \end{aligned} \tag{5.1}$$

The architecture $\mathcal{A}$ is usually designed by researchers and fixed during minimizing Eq. 5.1, and thus the solution $\mathcal{A},\theta_{\mathcal{A}}$ will always meet the resource constraint. When $\mathcal{A}$ is fixed, $\theta_{\mathcal{A}}$ found by standard gradient-based optimizers may have neurons (*i.e.* channels) that have little effects

on the average loss, removing which will save resources while maintaining good performance. In other words, $\theta_{\mathcal{A}}$ may not fully utilize all the resources available in $\mathcal{A}$. Let $\mathcal{U}(\theta_{\mathcal{A}})$ denote the computational cost based on $\theta_{\mathcal{A}}$'s actual utilization of the computational resource of $\mathcal{A}$, which can be measured by removing dead neurons which have little effect on the output. Clearly, $\mathcal{U}(\theta_{\mathcal{A}}) \leq c(\mathcal{A})$. As previous work suggests [181], the utilization ratio $r(\theta_{\mathcal{A}}) = \mathcal{U}(\theta_{\mathcal{A}})/c(\mathcal{A})$ trained by standard SGD can be as low as $11.5\%$.

The low utilization motivates the research on network pruning [181], [300], *i.e.*, extracting the effective subnet $\mathcal{A}'$ from $\mathcal{A}$ such that $c(\theta_{\mathcal{A}'}) = \mathcal{U}(\theta_{\mathcal{A}})$. Although the utilization ratio $r(\theta_{\mathcal{A}'})$ is high, this is the opposite to our problem because it tries to narrow the difference between $c(\mathcal{A})$ and $\mathcal{U}(\mathcal{A})$ by moving $c(\mathcal{A})$ towards $\mathcal{U}(\mathcal{A})$. By contrast, our objective is to design an optimization procedure $\mathcal{P}$ which enables us to find parameters $\theta_{\mathcal{A}} = \mathcal{P}(\mathcal{A}, \mathcal{L}, \mathcal{D})$ with a high $r(\theta_{\mathcal{A}})$. In other words, we are trying to move $\mathcal{U}(\mathcal{A})$ towards $c(\mathcal{A})$, which maximizes the real utilization of the constraint $\mathcal{C}$.

There are many reasons for the low utilization ratio $r(\theta_{\mathcal{A}})$. One is bad initialization [80], which can be alleviated by parameter reinitialization for the spare resource. Another one is inefficient resource allocation [95], *e.g.*, the numbers of channels or the depths of blocks may not be configured properly to meet their real needs. Unlike the previous methods [95], [175] which search architectures by training a lot of networks, we aim to design an optimizer that trains *one* network only *once* and includes both resource *reinitialization* and *reallocation* for maximizing resource utilization.

In this chapter, we propose an optimization method named Neural Rejuvenation (NR) for enhancing resource utilization during training. Our method is intuitive and simple. During training, as some neurons may be found to be useless (*i.e.* have little effect on the output), we revive them with new initialization and allocate them to the places they are needed the most. From a neuroscience perspective, this is to rejuvenate dead neurons by bringing them back to functional use [65] – hence the name. The challenges of Neural Rejuvenation are also clear. Firstly, we need a real-time resource utilization monitor. Secondly, when we rejuvenate dead

neurons, we need to know how to reinitialize them and where to place them. Lastly, after dead neuron rejuvenation, survived neurons ($\mathcal{S}$ neurons) and rejuvenated neurons ($\mathcal{R}$ neurons) are mixed up, and how to train networks with both of them present is unclear.

Our solution is a plug-and-play optimizer, the codes of which are public. Under the hood, it is built on standard gradient-based optimizers, but with additional functions including real-time resource utilization monitoring, dead neuron rejuvenation, and new training schemes designed for networks with mixed types of neurons. We introduce these components as below.

**Resource utilization monitoring.** Similar to [181], [300], we use the activation scales of neurons to identify utilized and spare computational resource, and calculate a real-time utilization ratio $r(\theta_{\mathcal{A}})$ during training. An event will be triggered if $r(\theta_{\mathcal{A}})$ is below a threshold $T_r$, and the procedure of dead neuron rejuvenation will take control before the next step of training, after which $r(\theta_{\mathcal{A}})$ will go back to $1$.

**Dead neuron rejuvenation.** This component rejuvenates the dead neurons by collecting the unused resources and putting them back in $\mathcal{A}$. Similar to MorphNet [95], more spare resources are allocated to the layers with more $\mathcal{S}$ neurons. However, unlike MorphNet [95] which trains the whole network again from scratch after the rearrangement, we only reinitialize the dead neurons and then continue training. By taking the advantages of dead neuron reinitialization [80] and our training schemes, our optimizer is able to train *one* model only *once* and outperform the optimal network found by MorphNet [95] from lots of architectures.

**Training with mixed neural types.** After dead neuron rejuvenation, each layer will have two types of neurons: $\mathcal{S}$ and $\mathcal{R}$ neurons. We propose two novel training schemes for different cases when training networks with mixed types of neurons. The first one is to remove the cross-connections between $\mathcal{S}$ and $\mathcal{R}$ neurons, and the second one is to use cross-attention between them to increase the network capacity. Sec. 5.3.3 presents the detailed discussions.

We evaluate Neural Rejuvenation on two common image recognition benchmarks, *i.e.* CIFAR-10/100 [141] and ImageNet [230] and show that it outperforms the baseline optimizer by

a very large margin. For example, we lower the top-1 error of ResNet-50 [108] on ImageNet by 1.51%, and by 1.82% for MobileNet-0.25 [116] while maintaining their FLOPs. On CIFAR where we rejuvenate the resources to the half of the constraint and compare with the previous state-of-the-art compression method [181], we outperform it by up to 0.87% on CIFAR-10 and 3.39% on CIFAR-100.

## 5.2 Related Work

### 5.2.1 Efficiency of Neural Networks

It is widely recognized that deep neural networks are over-parameterized [10], [59] to win the filter lottery tickets [80]. This efficiency issue is addressed by many methods, including weight quantization [54], [223], low-rank approximation [59], [152], knowledge distillation [112], [294] and network pruning [102], [106], [155], [161], [181], [194], [300], [304]. The most related method is network pruning, which finds the subnet that affects the outputs the most. Network pruning has several research directions, such as weight pruning, structural pruning, *etc*. Weight pruning focuses on individual weights [101], [102], [106], [155], but requires dedicated hardware and software implementations to achieve compression and acceleration [100]. Structural pruning identifies channels and layers to remove from the architecture, thus is able to directly achieve speedup without the need for specialized implementations [2], [111], [153], [186], [194], [273], [317]. Following [181], [300], we encourage channel sparsity by imposing penalty terms to the scaling factors.

Different from these previous methods, Neural Rejuvenation studies the efficiency issue from a new angle: we aim to directly maximize the utilization by reusing spare computational resources. As an analogy in the context of the lottery hypothesis [80], Neural Rejuvenation is like getting a refund for the useless tickets and then buying new ones.

### 5.2.2 Cross Attention

In this work, we propose to use cross attention to increase the capacity of the networks without introducing additional costs. This is motivated by adding second-order transform [93], [129], [269] on multi-branch networks [108], [120], [246], [247]. Instead of using a geometric mean as in [269], we propose to use cross attention [103], [158] as the second-order term to increase the capacity. Attention models have been widely used in deep neural networks for a variety of vision and language tasks, such as object detection [11], [193], machine translation [13], visual question answering [38], [288], image captioning [290], *etc*. Unlike the previous attention models, our method uses one group of channels to generate attentions for the other channels. Moreover, our attention model is mainly used to increase capacity.

### 5.2.3 Architecture Search

Our objective formulated by Eq. 5.1 is similar to neural architecture search which approaches the problem by searching architecture $\mathcal{A}$ in a pre-defined space, and thus they need to train a lot of networks to find the optimal architecture. For example, NAS [330] uses reinforcement learning to find the architecture, [331] extends it by using a more structured search space, and [175] improves the search efficiency by progressively finding architectures. But their computational costs are very high, *e.g.*, [331] uses 2000 GPU days. There are more methods focusing on the search problem [16], [29], [69], [191], [209], [225], [316]. Different from architecture search, Neural Rejuvenation does not search $\mathcal{A}$ which requires hundreds of thousands of models to train, although it does change the architecture a little bit. Instead, our method is an optimization technique that trains models in just one training pass. The closest method is MorphNet [95] in that we both use linearly expanding techniques to find resource arrangement. Yet, it still needs multiple training passes and does not rejuvenate dead neurons nor reuse partially-trained filters. We show direct comparisons with it and outperform it by a large margin.

### 5.2.4 Parameter Reinitialization

Parameter reinitialization is a common strategy in optimization to avoid useless computations and improve performances. For example, during the k-means optimization, empty clusters are automatically reassigned, and big clusters are encouraged to split into small clusters [35], [127], [128], [291]. Our method is reminiscent of this in that it also detects unsatisfactory components and reinitializes them so that they can better fit the tasks.

## 5.3 Neural Rejuvenation

Algorithm 3 presents a basic framework of Neural Rejuvenation which adds two new modules: resource utilization monitoring (Step 6) and dead neuron rejuvenation (Step 7 and 8) to a standard SGD optimizer. The training schemes are not shown here, which will be discussed in Sec. 5.3.3. We periodically set the Neural Rejuvenation flag on with a pre-defined time interval to check the utilization and rejuvenate dead neurons when needed. In the following subsections, we will present how each component is implemented.

### 5.3.1 Resource Utilization Monitoring

#### 5.3.1.1 Liveliness of Neurons

We consider a convolutional neural network where every convolutional layer is followed by a batch normalization layer [124]. An affine transform layer with learnable parameters is also valid if batch normalization is not practical. For each batch-normalized convolutional layer, let $\mathcal{B} = \{u_1, ..., u_m\}$ be a mini-batch of values after the convolution. Then, its normalized output $\{v_1, ..., v_m\}$ is

$$\begin{aligned} v_i &= \gamma \cdot \frac{u_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta, \quad \forall i \in \{1, ..., m\} \\ \text{where } \mu_{\mathcal{B}} &= \frac{1}{m}\sum_{i=1}^{m} u_i \text{ and } \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(u_i - \mu_{\mathcal{B}})^2 \end{aligned} \tag{5.2}$$

**Algorithm 3:** SGD with Neural Rejuvenation

**Input :** Learning rate $\epsilon$, utilization threshold $T_r$, initial architecture $\mathcal{A}$ and $\theta_{\mathcal{A}}$, and resource constraint $\mathcal{C}$

**while** *stopping criterion not met*:
  Sample a minibatch $\{(x_1, y_1), ..., (x_m, y_m)\}$;
  Compute gradient $g \leftarrow \frac{1}{m} \nabla \sum_i \mathcal{L}(f(x_i; \mathcal{A}, \theta_{\mathcal{A}}), y_i)$;
  Apply update $\theta_{\mathcal{A}} = \theta_{\mathcal{A}} - \epsilon \cdot g$;
  **if** *neural rejuvenation flag is on*:
    Compute utilization ratio $r(\theta_{\mathcal{A}})$;
    **if** $r(\theta_{\mathcal{A}}) < T_r$:
      Rejuvenate dead neurons and obtain new $\mathcal{A}$ and $\theta_{\mathcal{A}}$ under resource constraint $\mathcal{C}$;
**return** Architecture $\mathcal{A}$ and its parameter $\theta_{\mathcal{A}}$;

Each neuron (*i.e.* channel) in the convolutional layer has its own learnable scaling parameter $\gamma$, which we use as an estimate of the liveliness of the corresponding neuron [181], [300]. As our experiments suggest, if a channel's scaling parameter $\gamma$ is less than $0.01 \times \gamma_{\max}$ where $\gamma_{\max}$ is the maximum $\gamma$ in the same batch-normalized convolution layer, removing it will have little effect on the output of $f$ and the loss $\mathcal{L}$. Therefore, in all experiments shown in this chapter, a neuron is considered dead if its scaling parameter $\gamma < 0.01 \times \gamma_{\max}$. Let $\mathcal{T}$ be the set of all the scaling parameters within the architecture $\mathcal{A}$. Similar to [181], we add a L1 penalty term on $\mathcal{T}$ in order to encourage neuron sparsity, *i.e.,* instead of the given loss function $\mathcal{L}$, we minimize the following loss

$$\mathcal{L}_\lambda = \mathcal{L}\big(f(x_i; \mathcal{A}, \theta_{\mathcal{A}}), y_i\big) + \lambda \sum_{\gamma \in \mathcal{T}} |\gamma| \tag{5.3}$$

where $\lambda$ is a hyper-parameter.

#### 5.3.1.2 Computing $r(\theta_{\mathcal{A}})$ by Feed-Forwarding

Here, we show how to compute the utilization ratio $r(\theta_{\mathcal{A}})$ based on the liveliness of the neurons in real time. We compute $r(\theta_{\mathcal{A}})$ by a separate feed-forwarding similar to that of function $f$. The computational cost of the feed-forwarding for $r(\theta_{\mathcal{A}})$ is negligible compared with that of $f$. We first rewrite the function $f$:

$$f(x) = (f_l \circ f_{l-1} \circ ... \circ f_1)(x) \tag{5.4}$$

where $f_i$ is the i-th layer of the architecture $\mathcal{A}$. When computing $r(\theta_{\mathcal{A}})$, instead of passing the output of a layer computed from $x$ to the next layer as input, each layer $f_i$ will send a binary mask indicating the liveliness of its neurons. Let $M_i^{\text{in}}$ denote the binary mask for the input neurons for layer $f_i$, and $M_i^{\text{out}}$ denote the binary mask for its own neurons. Then, the effective number of parameters of $f_i$ is $||M_i^{\text{in}}||_1 \cdot ||M_i^{\text{out}}||_1 \cdot K_w \cdot K_h$, if $f_i$ is a convolutional layer with 1 group and no bias, and its computational cost is computed by $||M_i^{\text{in}}||_1 \cdot ||M_i^{\text{out}}||_1 \cdot K_w \cdot K_h \cdot O_w \cdot O_h$ following [108]. Here, $K_w$ and $K_h$ are the kernel size, and $O_w$ and $O_h$ are the output size. Note that the cost of $f$ is the sum of the costs of all layers $f_i$; therefore, we also pass the effective computational cost and the original cost in feed-forwarding. After that, we are able to compute $U(\theta_{\mathcal{A}})$ and $c(\theta_{\mathcal{A}})$, and consequently $r(\theta_{\mathcal{A}})$. During the computation of $r(\theta_{\mathcal{A}})$, each layer will also keep a copy of the liveliness of the neurons of its previous layer. This information is used in the step of dead neural rejuvenation after $r(\theta_{\mathcal{A}}) < T_r$ is met. It also records the values of the scaling parameter $\gamma$ of the input neurons. This is used for neural rescaling which is discussed in Sec. 5.3.2.1.

#### 5.3.1.3 Adaptive Penalty Coefficient $\lambda$

The utilization ratio $r(\theta_{\mathcal{A}})$ will depend on the value of the sparsity coefficient $\lambda$ as a larger $\lambda$ tends to result in a sparser network. When $\lambda = 0$, all neurons will probably stay alive as we have a tough threshold $0.01 \times \gamma_{\max}$. As a result, Step 7 and 8 of Algorithm 3 will never get executed and our optimizer is behaving as the standard one. When $\lambda$ goes larger, the real loss function $\mathcal{L}_\lambda$ we optimize will become far from the original loss $\mathcal{L}$. Consequently, the performance will be less unsatisfactory. Therefore, choosing a proper $\lambda$ is critical for our problem, and we would like it to be automatic and optimized to the task and the architecture.

In Neural Rejuvenation, the value of $\lambda$ is dynamically determined by the trend of the utilization ratio $r(\theta_{\mathcal{A}})$. Specifically, when the neural rejuvenation flag is on, we keep a record of the utilization ratio $r(\theta_{\mathcal{A}})^t$ after training for $t$ iterations. After $\Delta t$ iterations , we compare the current ratio $r(\theta_{\mathcal{A}})^t$ with the previous one $r(\theta_{\mathcal{A}})^{t-\Delta t}$. If $r(\theta_{\mathcal{A}})^t < r(\theta_{\mathcal{A}})^{t-\Delta t} - \Delta r$, we keep the

current $\lambda$; otherwise, we increase $\lambda$ by $\Delta\lambda$. Here, $\Delta t$, $\Delta r$ and $\Delta\lambda$ are hyper-parameters. $\lambda$ is initialized with $0$. After Step 8 gets executed, $\lambda$ is set back to $0$.

It is beneficial to set $\lambda$ in the above way rather than having a fixed value throughout the training. Firstly, different tasks and architectures may require different values of $\lambda$. The above strategy frees us from manually selecting one based on trial and error. Secondly, the number of iterations needed to enter Step 8 is bounded. This is because after $\lambda$ gets large enough, each $\Delta t$ will decrease the utilization ratio by at least $\Delta r$. Hence, the number of iterations to reach $T_r$ is bounded by $(1 - T_r)/\Delta r + O(1)$. In a word, this strategy automatically finds the value of $\lambda$, and guarantees that the condition $r(\theta_{\mathcal{A}}) < T_r$ will be met in a bounded number of training iterations.

### 5.3.2 Dead Neuron Rejuvenation

After detecting the liveliness of the neurons and the condition $r(\theta_{\mathcal{A}}) < T_r$ is met, we proceed to Step 8 of Algorithm 3. Here, our objective is to rejuvenate the dead neurons and reallocate those rejuvenated neurons to the places they are needed the most under the resource constraint $\mathcal{C}$. There are three major steps in dead neuron rejuvenation. We present them in order as follows.

#### 5.3.2.1 Resource Reallocation

The first step is to reallocate the computational resource saved by removing all the dead neurons. The removal reduces the computational cost from $c(\mathcal{A})$ to $U(\theta_{\mathcal{A}})$; therefore, there is $c(\mathcal{A}) - U(\theta_{\mathcal{A}})$ available resource to reallocate. The main question is where to add this free resource back in $\mathcal{A}$. Let $w_i$ denote the number of output channels of layer $f_i$ in $f$, and $w_i$ is reduced to $w_i'$ by dead neuron removal. Let $\mathcal{A}'$ denote the architecture after dead neuron removal with $w_i'$ output channels at layer $f_i$. Then, $c(\mathcal{A}') = U(\mathcal{A})$. To increase the computational cost of $\mathcal{A}'$ to the level of $\mathcal{A}$, our resource reallocation will linearly expand $w_i'' = \alpha \cdot w_i'$ by a shared expansion rate $\alpha$ across all the layers $f_i$, to build a new architecture $\mathcal{A}''$ with numbers of

channels $w_i''$. The assumption here is that if a layer has a higher ratio of living neurons, this layer needs more resources, *i.e.* more output channels; by contrast, if a layer has a lower ratio, this means that more than needed resources were allocated to it in $\mathcal{A}$. This assumption is modeled by having a shared linear expansion rate $\alpha$.

The resource reallocation used here is similar to the iterative squeeze-and-expand algorithm in MorphNet [95] for neural architecture search. The differences are also clear. Neural Rejuvenation models both dead neuron reinitialization, reallocation, and training schemes to train just one network only once, while MorphNet is only interested in the numbers of channels of each layer that are optimal when trained from scratch and finds it by training many networks.

#### 5.3.2.2 Parameter Reinitialization

The second step is to reinitialize the parameters of the reallocated neurons. Let $\mathcal{S}_{\text{in}}$ and $\mathcal{R}_{\text{in}}$ denote the input $S$ (survived) neurons and $R$ (rejuvenated) neurons, respectively, and $\mathcal{S}_{\text{out}}$ and $\mathcal{R}_{\text{out}}$ denote the output $S$ neurons and $R$ neurons, respectively. Then, the parameters $W$ can be divided into four groups: $W_{\mathcal{S}\to\mathcal{S}}$, $W_{\mathcal{S}\to\mathcal{R}}$, $W_{\mathcal{R}\to\mathcal{R}}$, $W_{\mathcal{R}\to\mathcal{S}}$, which correspond to the parameters from $\mathcal{S}_{\text{in}}$ to $\mathcal{S}_{\text{out}}$, from $\mathcal{S}_{\text{in}}$ to $\mathcal{R}_{\text{out}}$, from $\mathcal{R}_{\text{in}}$ to $\mathcal{R}_{\text{out}}$ and from $\mathcal{R}_{\text{in}}$ to $\mathcal{S}_{\text{out}}$, respectively. During reinitialization, the parameters $W_{\mathcal{S}\to\mathcal{S}}$ are kept since they survive the dead neuron test. The parameters $W_{\mathcal{R}\to\mathcal{R}}$ are randomly initialized and their scaling parameters $\gamma$'s are restored to the initial level. In order for the $\mathcal{S}$ neurons to keep their mapping functions after the rejuvenation, $W_{\mathcal{R}\to\mathcal{S}}$ is set to $0$. We also set $W_{\mathcal{S}\to\mathcal{R}}$ to $0$ as this initialization does not affect the performances as the experiments suggest.

#### 5.3.2.3 Neural Rescaling

Recall that in order to encourage the sparsity of the neurons, all neurons receive the same amount of penalty. This means that not only the dead neurons have small scaling values, some $\mathcal{S}$ neurons also have scaling values that are very small compared with $\gamma_{\max}$ of the corresponding layers. As experiments in Sec. 5.4.2 show, this is harmful to gradient-based training. Our

solution is to rescale those neurons to the initial level, *i.e.*, $|\gamma_i'| = \max\{|\gamma_i|, \gamma_0\}\ \forall i$, where $\gamma_0$ is the initial value for $\gamma$. We do not change the sign of $\gamma$. Note that rescaling takes all neurons into consideration, including $\mathcal{S}$ neurons with large scaling values ($|\gamma| \geq |\gamma_0|$), $\mathcal{S}$ neurons with small scaling values ($|\gamma| < |\gamma_0|$) and dead neurons ($|\gamma| \approx 0$). After neural rescaling, we adjust the parameters to restore the original mappings. For $\mathcal{S}$ neurons, let $s_i = \gamma_i'/\gamma_i$. In order for $\mathcal{S}$ neurons to keep their original mapping functions, we divide the parameters that use them by $s_i$. Experiments show that this leads to performance improvements.

### 5.3.3 Training with Mixed Types of Neurons

Let us now focus on each individual layer. After neural rejuvenation, each layer will have two types of input neurons, $\mathcal{S}_{\text{in}}$ and $\mathcal{R}_{\text{in}}$, and two types of output neurons, $\mathcal{S}_{\text{out}}$ and $\mathcal{R}_{\text{out}}$. For simplicity, we also use them to denote the features of the corresponding neurons. Then, by the definition of convolution, we have

$$\begin{aligned} \mathcal{S}_{\text{out}} &= W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} + W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}} \\ \mathcal{R}_{\text{out}} &= W_{\mathcal{S}\to\mathcal{R}} * \mathcal{S}_{\text{in}} + W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}} \end{aligned} \tag{5.5}$$

where $*$ denote the convolution operation. $W_{\mathcal{R}\to\mathcal{S}}$ is set to $0$ in reinitialization; therefore, $\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$ initially, which keeps the original mappings between $\mathcal{S}_{\text{in}}$ and $\mathcal{S}_{\text{out}}$. In this subsection, we discuss how to train $W$.

The training of $W$ depends on how much the network needs the additional capacity brought by the rejuvenated neurons to fit the data. When $\mathcal{S}$ neurons do not need this additional capacity at all, adding $\mathcal{R}$ neurons by Eq. 5.5 may not help because $\mathcal{S}$ neurons alone are already able to fit the data well. As a result, changing training scheme is necessary in this case in order to utilize the additional capacity. However, when $\mathcal{S}$ neurons alone have difficulties fitting the data, the additional capacity provided by $\mathcal{R}$ neurons will ease the training. They were found to be useless previously either because of improper initialization or inefficient resource arrangement, but now are reinitialized and rearranged. We present the detailed discussions as below.

#### 5.3.3.1 When $\mathcal{S}$ Does Not Need $\mathcal{R}$

Here, we consider the situation where the network capacity is bigger than necessary, and $\mathcal{S}$ neurons alone are able to fit the training data well. An example is training networks on CIFAR [141], where most of the modern architectures can reach $99.0\%$ training accuracy. When adding $\mathcal{R}$ neurons into the architecture as in Eq. 5.5, since $\mathcal{S}$ neurons have already been trained to fit the data well, the gradient back-propagated from the loss will not encourage any great changes on the local mapping $(\mathcal{S}_{\text{in}}, \mathcal{R}_{\text{in}}) \rightarrow (\mathcal{S}_{\text{out}}, \mathcal{R}_{\text{out}})$. Therefore, keep modeling the computation as Eq. 5.5 may result in $\mathcal{R}_{\text{in}}$ neurons being dead soon and $\mathcal{R}_{\text{out}}$ producing redundant features.

The cause of the above problem is the existence of cross-connections between $\mathcal{R}$ neurons and $\mathcal{S}$ neurons, which provides short-cuts to $\mathcal{R}$. If we completely remove them, *i.e.*,

$$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\rightarrow\mathcal{S}} * \mathcal{S}_{\text{in}} \quad \mathcal{R}_{\text{out}} = W_{\mathcal{R}\rightarrow\mathcal{R}} * \mathcal{R}_{\text{in}} \tag{5.6}$$

then $\mathcal{R}$ neurons are forced to learn features that are new and ideally different. We use NR-CR to denote Neural Rejuvenation with cross-connections removed.

#### 5.3.3.2 When $\mathcal{S}$ Needs $\mathcal{R}$

Here, we assume that the capacity of $\mathcal{S}$ alone is not enough for fitting the training data. One example is training small networks on ImageNet dataset [230]. In this case, it is desirable to keep the cross-connections to increase the capacity. Experiments in Sec. 5.4.2 compare the performances of a simplified VGG network [239] on ImageNet, and show that Neural Rejuvenation with cross-connections kept and removed both improve the accuracies, but keeping cross-connections improves more.

#### 5.3.3.3 Cross-Attention Between $\mathcal{S}$ and $\mathcal{R}$

We continue the discussion where we assume $\mathcal{S}$ needs the capacity of $\mathcal{R}$ and we keep the cross-connections. Then according to Eq. 5.5, the outputs from $\mathcal{S}_{\text{in}}$ and $\mathcal{R}_{\text{in}}$ are added up for

$\mathcal{S}_{\text{out}}$, *i.e.*

$$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} + W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}} \tag{5.7}$$

Since the assumption here is that the model capacity is insufficient for fitting the training data, it would be better if we can increase the capacity not only by rejuvenating dead neurons, but also by changing Eq. 5.7 to add more capacity without using any more parameters nor resulting in substantial increases of computations (if any) compared with the convolution operation itself. As $W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$ is fixed, we focus on $W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}}$. One way to increase capacity is to use second-order response transform [269]. The original second-order response transform is defined on residual learning [108] by adding a geometric mean, *i.e.*

$$y = x + F(x) \Rightarrow y = x + F(x) + \sqrt{x \cdot F(x)} \tag{5.8}$$

For our problem, although Eq. 5.7 does not have residual connections, the outputs $W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}$ and $W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}}$ are added up as in residual learning; therefore, we can add a similar response transform to Eq. 5.7. Instead of adding a geometric mean which causes training instability [269], we propose to use cross attentions as shown in Eq. 5.9.

$$\mathcal{S}_{\text{out}} = W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}} + 2 \cdot \sigma(W_{\mathcal{S}\to\mathcal{S}} * \mathcal{S}_{\text{in}}) W_{\mathcal{R}\to\mathcal{S}} * \mathcal{R}_{\text{in}} \tag{5.9}$$

Here, $\sigma(\cdot)$ denotes the Sigmoid function. Symmetrically, we add cross attentions to the output of $\mathcal{R}_{\text{out}}$, *i.e.*

$$\mathcal{R}_{\text{out}} = W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}} + 2 \cdot \sigma(W_{\mathcal{R}\to\mathcal{R}} * \mathcal{R}_{\text{in}}) W_{\mathcal{S}\to\mathcal{R}} * \mathcal{S}_{\text{in}} \tag{5.10}$$

We use NR-CA to denote NR with cross attentions.

## 5.4 Experiments

In this section, we will show the experimental results that support our previous discussions, and present the improvements of Neural Rejuvenation on a variety of architectures.

### 5.4.1 Resource Utilization

We show the resource utilization of training ResNet-50 and ResNet-101 on ImageNet in Figure 5-1 when the sparsity term is added to the loss. In the figure, we show the plots of the parameter utilization and validation accuracy of the models with respect to the number of training epochs. Training such a model usually takes 90 epochs when the batch size is 256 or 100 epochs when the batch size is 128 [120]. In all the experiments, the sparsity coefficient $\lambda$ is initialized with 0, $\Delta t$ is set to one epoch, $\Delta r = 0.01$ and $\Delta\lambda = 5 \times 10^{-5}$. $T_r$ is set to $0.5$ unless otherwise stated.

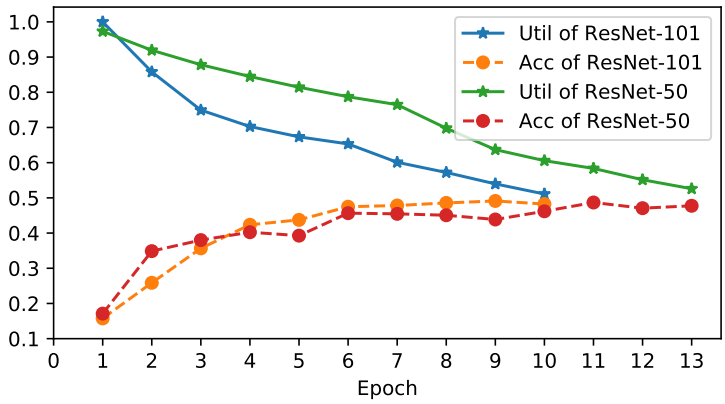



**Figure 5-1.** Parameter utilization and validation accuracy of ResNet-50 and ResNet-101 trained on ImageNet from scratch.

Fig. 5-1 shows two typical examples that convey the following important messages. (1) Training on large-scale dataset such as ImageNet cannot avoid the waste of computational resources; therefore, our work is also valid for large-scale training. (2) It is easier to find dead neurons in larger models than in smaller models. This is consistent with our intuition that larger models increase the capacity and the risk of more resource wastes. (3) It does not take too long to reach the utilization threshold at 0.5. 10 epochs are enough for saving half of the resources for ResNet-101.

For ImageNet training, we set the neural rejuvenation flag on only for the first $30$ epochs where the learning rate is $0.1$. Since it usually takes 10-20 epochs for $r(\theta_{\mathcal{A}})$ to reach $T_r = 0.5$, there will be about 1 to 2 times that Step 8 in Algorithm 3 will get executed. To simplify the

experiments, we only do one time of neural rejuvenation on ImageNet and reset the epoch counter to 0 afterward. The training time with neural rejuvenation thus will be a little longer than the original training, but the increase will be less than $20\%$ and experiments show that it is definitely worth it. For unlimited training time, Sec. 5.4.4.2 shows the performances on CIFAR with multiple times of Neural Rejuvenation.

### 5.4.2 Ablation Study on Neural Rejuvenation

To provide better understandings of Neural Rejuvenation applied on training deep networks, we present an ablation study shown in Table 5-I, which demonstrates the results of Neural Rejuvenation with different variations.

| Method | Top-1 | Top-5 | Method | Top-1 | Top-5 |
|---|---|---|---|---|---|
| BL | 32.13 | 11.97 | BL-CA | 31.58 | 11.46 |
| NR-CR | 31.40 | 11.53 | NR-FS | 31.26 | 11.37 |
| NR | 30.74 | 10.94 | NR-BR | 30.31 | 10.67 |
| NR-CA | 30.28 | 10.88 | NR-CA-BR | 29.98 | 10.58 |

**Table 5-I.** Error rates of a simplified VGG-19 on ImageNet with $T_r = 0.25$ while maintaining the number of parameters. BL: baseline. BL-CA: baseline with cross attentions. NR-CR: Neural Rejuvenation with cross-connections removed. NR-FS: training $\mathcal{A}$ found by NR from scratch. NR: Neural Rejuvenation with cross-connections. NR-BR: Neural Rejuvenation with neural rescaling. NR-CA: Neural Rejuvenation with cross attentions. NR-CA-BR: Neural Rejuvenation with cross attentions and neural rescaling.

The network is a simplified VGG-19, which is trained on low-resolution images from ImageNet. The image size for training and testing is 128x128. We remove the last three fully-connected layers, and replace them with a global average pooling layer and one fully-connected layer. The resulting model has only $20.5$M parameters. To further accelerate training, we replace the first convolutional layer with that in ResNet [108]. By applying all the changes, we can train one model with 4 Titan Xp GPUs in less than one day, which is fast enough for the purpose of ablation study.

Clearly, such a simplified model does not have sufficient capacity for fitting ImageNet. As we have discussed in Sec. 5.3.3, it is better to keep the cross connections for increasing the model

| Architecture | Baseline | | | | NR Params | | | | NR FLOPs | | | | Relative |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Params | FLOPs | Top-1 | Top-5 | Params | FLOPs | Top-1 | Top-5 | Params | FLOPs | Top-1 | Top-5 | Gain |
| DenseNet-121 [120] | 7.92M | 2.83G | 25.32 | 7.88 | 8.22M | 3.13G | 24.50 | 7.49 | 7.28M | 2.73G | 24.78 | 7.56 | -3.24% |
| VGG-16 [239] | 37.7M | 15.3G | 24.26 | 7.32 | 36.4M | 23.5G | 23.11 | 6.69 | 21.5M | 15.3G | 23.71 | 7.01 | -4.74% |
| ResNet-18 [108] | 11.7M | 1.81G | 30.30 | 10.7 | 11.9M | 2.16G | 28.86 | 9.93 | 9.09M | 1.73G | 29.73 | 10.5 | -4.75% |
| ResNet-34 [108] | 21.8M | 3.66G | 26.61 | 8.68 | 21.9M | 3.77G | 25.77 | 8.10 | 20.4M | 3.56G | 25.45 | 8.04 | -4.35% |
| ResNet-50 [108] | 25.6M | 4.08G | 24.30 | 7.19 | 26.4M | 3.90G | 22.93 | 6.47 | 26.9M | 3.99G | 22.79 | 6.56 | -6.21% |
| ResNet-101 [108] | 44.5M | 7.80G | 22.44 | 6.21 | 46.6M | 6.96G | 21.22 | 5.76 | 50.2M | 7.51G | 20.98 | 5.69 | -6.50% |

**Table 5-II.** Error rates of deep neural networks on ImageNet validation set trained with and without Neural Rejuvenation. Each neural network has three sets of top-1 and top-5 error rates, which are baseline, Neural Rejuvenation with the number of parameters as the resource constraint (NR Params), and Neural Rejuvenation with FLOPs as resource constraint (NR FLOPs). The last column *Relative Gain* shows the best relative gain of top-1 error while maintaining either number of parameters or FLOPs.

capacity. As also demonstrated here, NR-CR improves the top-1 accuracy by $0.7\%$ than the baseline, but is $0.7\%$ behind NR where cross-connections are kept. We further show that cross attentions lower the top-1 error rates by roughly $0.5\%$, and neural rescaling further improves the accuracies. In the following experiments on ImageNet, we use NR-CA-BR for all the methods.

### 5.4.3 Results on ImageNet

Table 5-II shows the performance improvements on ImageNet dataset [230]. ImageNet dataset is a large-scale image classification dataset, which contains about 1.28 million color images for training and 50,000 for validation. Table 5-II lists some modern architectures which achieve very strong accuracies on such a challenging task. Previously, a lot of attention is paid to designing novel architectures that are more suitable for vision tasks. Our results show that in addition to architecture design and search, the current optimization technique still has a lot of room to improve. Our work focuses only on the utilization issues, but already achieves strong performance improvements.

Here, we briefly introduce the setting of the experiments for easy reproduction. All the models are trained with batch size 256 if the model can fit in the memory; otherwise, we set the batch size to 128. In total, we train the models for 90 epochs when the batch size is 256, and for 100 epochs if the batch size is 128. The learning rate is initialized as 0.1, and then divided

by 10 at the 31$^{st}$, 61$^{st}$, and 91$^{st}$ epoch.

For our task, we make the following changes to those state-of-the-art architectures. For VGG-16 [239], we add batch normalization layers after each convolutional layer and remove the last three fully-connected layers. After that, we add two convolutional layers that both output 4096 channels, in order to follow the original VGG-16 that has two fully-connected layers outputting the same amount of channels. After these two convolutional layers, we add a global average pooling layer, and a fully-connected layer that transforms the 4096 channels to 1000 channels for image classification. The resulting model has a fewer number of parameters (138M to 37.7M), but with a much lower top-1 error rate (27 to 24.26). All the VGG-16 layers receive the sparsity penalty. For ResNet [108], all the convolutional layers except the ones that are added back to the main stream are taken into the consideration for neural rejuvenation. For DenseNet [120], due to the GPU memory and speed issue, we are only able to run DenseNet with 121 layers. We change it from pre-activation [109] to post-activation [108] to follow our assumption that each convolutional layer is directly followed by a batch normalization layer. This change yields a similar accuracy to the original one.

A quick observation of our results is that the models with stronger capacities actually have better improvements from Neural Rejuvenation. This is consistent with our discussion in Sec. 5.3.3 and the observation in Sec. 5.4.1. For large-scale tasks, the model capacity is important and larger models are more likely to waste more resources. Therefore, rejuvenating dead neurons in large models will improve more than doing that in small models where the resources are better utilized. In all models, DenseNet-121 is the hardest to find dead neurons, and thus has the smallest improvements. This may explain the model compactness discussed in their paper [120]. Moreover, VGG-16 with NR achieves 23.71% top-1 error with just 21.5M parameters, far better than [181] which achieves 36.66% top-1 error with 23.2M.

Next, we show experiments on MobileNet-0.5 and 0.25 in Table 5-III. They are not included in Table 5-II because their image size is 128x128 and the learning rate follows the cosine learning rate schedule starting from 0.1 [218]. MobileNet is designed for platforms with low

| Architecture | BL [95] | MN [95] | BL* | NR |
|---|---|---|---|---|
| MobileNet-0.50 | 42.9 | 41.9 | 41.77 | 40.12 |
| MobileNet-0.25 | 55.2 | 54.1 | 53.76 | 51.94 |

**Table 5-III.** Top-1 error rates of MobileNet [116] on ImageNet. The image size is 128x128 for both training and testing. The FLOPs are maintained in all the methods. BL: the baseline performances reported in [95], MN: MorphNet [95], BL*: our implementation of the baseline, and NR: Neural Rejuvenation.

| Architecture | Baseline | | Network Slimming [181] | | Neural Rejuvenation | |
|---|---|---|---|---|---|---|
| | C10 (Params) | C100 (Params) | C10 (Params) | C100 (Params) | C10 (Params) | C100 (Params) |
| VGG-19 [239] | 5.44 (20.04M) | 23.11 (20.08M) | 5.06 (10.07M) | 24.92 (10.32M) | 4.19 (9.99M) | 21.53 (10.04M) |
| ResNet-164 [108] | 6.11 (1.70M) | 28.86 (1.73M) | 5.65 (0.94M) | 25.61 (0.96M) | 5.13 (0.88M) | 23.84 (0.92M) |
| DenseNet-100-40 [120] | 3.64 (8.27M) | 19.85 (8.37M) | 3.75 (4.36M) | 19.29 (4.65M) | 3.40 (4.12M) | 18.59 (4.31M) |

**Table 5-IV.** Neural Rejuvenation for model compression on CIFAR [141]. In the experiments for ImageNet, the computational resources are kept when rejuvenating dead neurons. But here, we set the resource target of neural rejuvenation to half of the original usage. Then, our Neural Rejuvenation becomes a model compressing method, and thus can be compared with the state-of-the-art pruning method [181].

computational resources. Our NR outperforms the previous method [95] and shows very strong improvements.

### 5.4.4 Results on CIFAR

The experiments on CIFAR have two parts. The first part is to use Neural Rejuvenation as a model compression method to compare with the previous state-of-the-arts when the model sizes are halved. The results are shown in Table 5-IV. In the second part, we show the performances in Table 5-V where we do Neural Rejuvenation multiple times.

#### 5.4.4.1 Model Compression

Table 5-IV shows the performance comparisons on CIFAR-10/100 datasets [141]. CIFAR dataset is a small dataset, with 50,000 training images and 10,000 test images. Unlike our experiments on ImageNet, here, we do not rejuvenate dead neurons to utilize all the available computational resource; instead, we set the resource target to $0.5 \times \mathcal{C}$ where $\mathcal{C}$ is the original

resource constraint. In practice, this is done by setting $T_r = 0.25$ and rejuvenating the models to the level of $0.5 \times \mathcal{C}$. As a result, Neural Rejuvenation ends up training a model with only a half of the parameters, which can be compared with the previous state-of-the-art network pruning method [181].

#### 5.4.4.2 Multiple NR

Table 5-V shows the performances of VGG-19 tested on CIFAR datasets without limiting the times of Neural Rejuvenation. The improvement trends are clear when the number of Neural Rejuvenation increases. The relative gains are $33.5\%$ for CIFAR-10 and $13.8\%$ for CIFAR-100.

| # of NR | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| C10 | 5.44 | 4.19 | 4.03 | 3.79 | 3.69 | 3.62 |
| C100 | 23.11 | 21.53 | 20.47 | 19.91 | — | — |

**Table 5-V.** Error rates of VGG-19 on CIFAR-10 (C10) and CIFAR-100 (C100) with different times of Neural Rejuvenation while maintaining the number of parameters.

Here, we introduce the detailed settings of the experiments. For VGG-19, we make the following changes because the original architecture is not designed for CIFAR. First, we remove all the fully-connected layers and add a global average pooling layer after the convolutional layers which is then followed by a fully-connected layer that produces the final outputs. Then, we remove the original 4 max-pooling layers and add 2 max-pooling layers after the 4$^{\text{th}}$ and the 10$^{\text{th}}$ convolutional layers for downsampling. These changes adapt the original architecture to CIFAR, and the baseline error rates become lower, *e.g.* from 6.66 to 5.44 on CIFAR-10 and from 28.05 to 23.11 on CIFAR-100. We make the same changes to DenseNet as for ImageNet. For ResNet-164 with bottleneck blocks, similar to our settings on ImageNet, we only consider the neurons that are not on the mainstream of the network for Neural Rejuvenation. Our method is NR-CR, which removes all the cross-connections. Table 5-IV shows that our Neural Rejuvenation can be used for training small models as well. Table 5-V presents the potential of VGG-19 when trained with multiple times of Neural Rejuvenation. While maintaining

the number of parameters, Neural Rejuvenation improves the performances by a very large margin.

## 5.5 Conclusion

In this chapter, we study the problem of improving the training of deep neural networks by enhancing computational resource utilization. This problem is motivated by two observations on deep network training, (1) more computational resources usually lead to better performances, and (2) the resource utilization of models trained by standard optimizers may be unsatisfactory. Therefore, we study the problem of maximizing resource utilization. To this end, we propose a novel method named Neural Rejuvenation, which rejuvenates dead neurons during training by reallocating and reinitializing them. Neural rejuvenation is composed of three components: resource utilization monitoring, dead neuron rejuvenation, and training schemes for networks with mixed types of neurons. These components detect the liveliness of neurons in real time, rejuvenate dead ones when needed and provide different training strategies when the networks have mixed types of neurons. We test neural rejuvenation on the challenging datasets CIFAR and ImageNet, and show that our method can improve a variety of state-of-the-art network architectures while maintaining either their numbers of parameters or the loads of computations. Moreover, when we target the architecture to a lower computational cost, Neural Rejuvenation can be used for model compression, which also shows better performances than the previous state-of-the-arts. In conclusion, Neural Rejuvenation is an optimization technique with a focus on resource utilization, which improves the training of deep neural networks by enhancing the utilization.

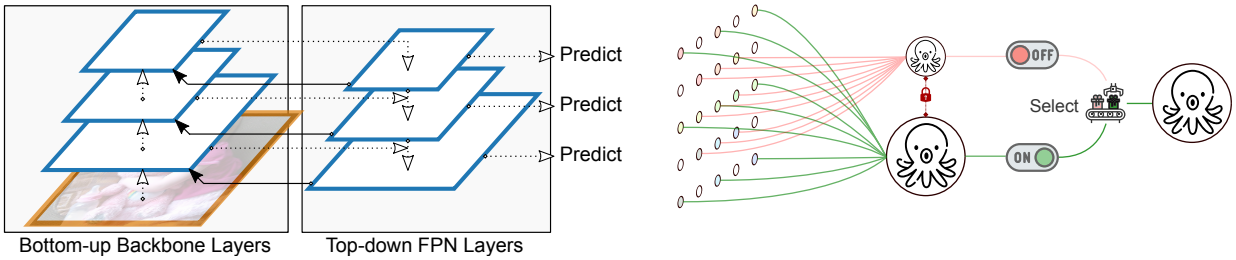
# Chapter 6

# DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution

Many modern object detectors demonstrate outstanding performances by using the mechanism of looking and thinking twice. In this chapter, we explore this mechanism in the backbone design for object detection. At the macro level, we propose Recursive Feature Pyramid, which incorporates extra feedback connections from Feature Pyramid Networks into the bottom-up backbone layers. At the micro level, we propose Switchable Atrous Convolution, which convolves the features with different atrous rates and gathers the results using switch functions. Combining them results in DetectoRS, which significantly improves the performances of object detection. On COCO test-dev, DetectoRS achieves state-of-the-art 55.7% box AP for object detection, 48.5% mask AP for instance segmentation, and 50.0% PQ for panoptic segmentation.

## 6.1 Introduction

To detect objects, human visual perception selectively enhances and suppresses neuron activation by passing high-level semantic information through feedback connections [18], [60], [61]. Inspired by the human vision system, the mechanism of *looking and thinking twice* has been instantiated in computer vision, and demonstrated outstanding performance [31], [32], [228]. Many popular two-stage object detectors, *e.g.*, Faster R-CNN [228], output object




**(a)** Macro Design: Recursive Feature Pyramid. **(b)** Micro Design: Switchable Atrous Convolution.

**Figure 6-1.** (a) Our Recursive Feature Pyramid adds feedback connections (solid lines) from the top-down FPN layers to the bottom-up backbone layers to look at the image twice or more. (b) Our Switchable Atrous Convolution looks twice at the input features with different atrous rates and the outputs are combined together by soft switches.

proposals first, based on which regional features are then extracted to detect objects. Following the same direction, Cascade R-CNN [31] develops a multi-stage detector, where subsequent detector heads are trained with more selective examples. The success of this design philosophy motivates us to explore it in the neural network backbone design for object detection. In particular, we deploy the mechanism at both the macro and micro levels, resulting in our proposed DetectoRS which significantly improves the performance of the state-of-art object detector HTC [36] by a great margin while a similar inference speed is maintained, as shown in Tab. 6-I.

At the macro level, our proposed Recursive Feature Pyramid (RFP) builds on top of the Feature Pyramid Networks (FPN) [171] by incorporating extra feedback connections from the FPN layers into the bottom-up backbone layers, as illustrated in Fig. 6-1a. Unrolling the recursive structure to a sequential implementation, we obtain a backbone for object detector that looks at the images twice or more. Similar to the cascaded detector heads in Cascade R-CNN trained with more selective examples, our RFP recursively enhances FPN to generate increasingly powerful representations. Resembling Deeply-Supervised Nets [156], the feedback connections bring the features that directly receive gradients from the detector heads back to the low levels of the bottom-up backbone to speed up training and boost performance. Our proposed RFP implements a sequential design of *looking and thinking twice*, where the

| Method | Backbone | $AP_{box}$ | $AP_{mask}$ | FPS |
|---|---|---|---|---|
| HTC [36] | ResNet-50 | 43.6 | 38.5 | 4.3 |
| DetectoRS | ResNet-50 | 51.3 | 44.4 | 3.9 |

**Table 6-I.** A glimpse of the improvements of the box and mask AP by our DetectoRS on COCO `test-dev`.

bottom-up backbone and FPN are run multiple times with their output features dependent on those in the previous steps.

At the micro level, we propose Switchable Atrous Convolution (SAC), which convolves the same input feature with different atrous rates [42], [115], [205] and gathers the results using switch functions. Fig. 6-1b shows an illustration of the concept of SAC. The switch functions are spatially dependent, *i.e.*, each location of the feature map might have different switches to control the outputs of SAC. To use SAC in the detector, we convert all the standard 3x3 convolutional layers in the bottom-up backbone to SAC, which improves the detector performance by a large margin. Some previous methods adopt conditional convolution, *e.g.*, [165], [293], which also combines results of different convolutions as a single output. Unlike those methods whose architecture requires to be trained from scratch, SAC provides a mechanism to easily convert pretrained standard convolutional networks (*e.g.*, ImageNet-pretrained [230] checkpoints). Moreover, a new weight locking mechanism is used in SAC where the weights of different atrous convolutions are the same except for a trainable difference.

Combining RFP and SAC results in our DetectoRS. To demonstrate its effectiveness, we incorporate DetectoRS into the state-of-art HTC [36] on the challenging COCO dataset [173]. On COCO `test-dev`, we report box AP for object detection [71], mask AP for instance segmentation [105], and PQ for panoptic segmentation [136]. DetectoRS with ResNet-50 [108] as backbone significantly improves HTC [36] by 7.7% box AP and 5.9% mask AP. Additionally, equipping our DetectoRS with ResNeXt-101-64x4d [281] achieves state-of-the-art 55.7% box AP and 48.5% mask AP. Together with the stuff prediction from DeepLabv3+ [44] with Wide-ResNet-41 [40] as backbone, DetectoRS sets achieved 50.0% PQ for panoptic segmentation.

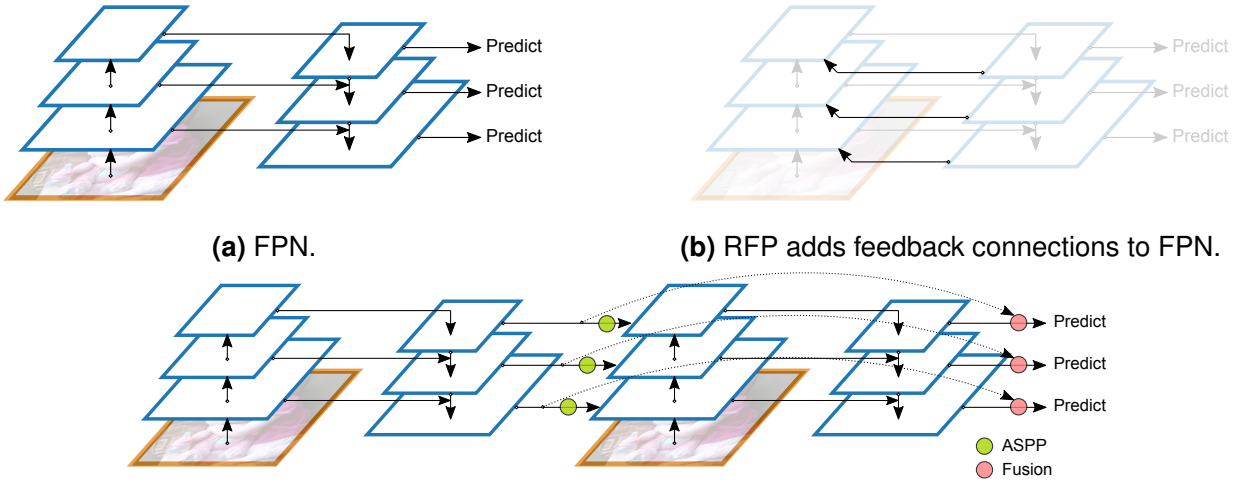
## 6.2 Related Works

**Object Detection**. There are two main categories of object detection methods: one-stage methods, *e.g.*, [162], [172], [178], [226], [236], [265], [313], [315], and multi-stage methods, *e.g.*, [31], [33], [36], [39], [89], [99], [107], [125], [228], [276]. Multi-stage detectors are usually more flexible and accurate but more complex than one-stage detectors. In this chapter, we use a multi-stage detector HTC [36] as our baseline and show comparisons with both categories.

**Multi-Scale Features**. Our Recursive Feature Pyramid is based on Feature Pyramid Networks (FPN) [171], an effective object detection system that exploits multi-scale features. Previously, many object detectors directly use the multi-scale features extracted from the backbone [30], [178], while FPN incorporates a top-down path to sequentially combine features at different scales. PANet [177] adds another bottom-up path on top of FPN. STDL [318] proposes to exploit cross-scale features by a scale-transfer module. G-FRNet [3] adds feedback with gating units. NAS-FPN [88] and Auto-FPN [287] use neural architecture search [330] to find the optimal FPN structure. EfficientDet [251] proposes to repeat a simple BiFPN layer. Unlike them, our proposed Recursive Feature Pyramid goes through the bottom-up backbone repeatedly to enrich the representation power of FPN. Additionally, we incorporate the Atrous Spatial Pyramid Pooling (ASPP) [43], [44] into FPN to enrich features, similar to the mini-DeepLab design in Seamless [211].

**Recursive Convolutional Network**. Many recursive methods have been proposed to address different types of computer vision problems, *e.g.*, [133], [168], [249]. Recently, a recursive method CBNet [180] is proposed for object detection, which cascades multiple backbones to output features as the input of FPN. By contrast, our RFP performs recursive computations with proposed ASPP-enriched FPN *included* along with effective fusion modules.

**Conditional Convolution** Conditional convolutional networks adopt dynamic kernels, widths, or depths, *e.g.*, [48], [165], [169], [176], [293], [303]. Unlike them, our proposed Switchable Atrous Convolution (SAC) allows an effective conversion mechanism from standard convolutions




**(a)** FPN.

**(b)** RFP adds feedback connections to FPN.

**(c)** Unrolling RFP to an example 2-step sequential implementation.

**Figure 6-2.** The architecture of Recursive Feature Pyramid (RFP). (a) Feature Pyramid Networks (FPN). (b) Our RFP incorporates feedback connections into FPN. (c) RFP unrolled to a 2-step sequential network.

to conditional convolutions without changing any pretrained models. SAC is thus a plug-and-play module for many pretrained backbones. Moreover, SAC uses global context information and a novel weight locking mechanism to make it more effective.

## 6.3 Recursive Feature Pyramid

### 6.3.1 Feature Pyramid Networks

This subsection provides the background of Feature Pyramid Networks (FPN). Let $\mathbf{B}_i$ denote the $i$-th stage of the bottom-up backbone, and $\mathbf{F}_i$ denote the $i$-th top-down FPN operation. The backbone equipped with FPN outputs a set of feature maps $\{\mathbf{f}_i \mid i = 1, ..., S\}$, where $S$ is the number of the stages. For example, $S = 3$ in Fig. 6-2a. $\forall i = 1, ..., S$, the output feature $\mathbf{f}_i$ is defined by

$$\mathbf{f}_i = \mathbf{F}_i\big(\mathbf{f}_{i+1}, \mathbf{x}_i\big), \quad \mathbf{x}_i = \mathbf{B}_i\big(\mathbf{x}_{i-1}\big), \tag{6.1}$$

where $\mathbf{x}_0$ is the input image and $\mathbf{f}_{S+1} = \mathbf{0}$. The object detector built on FPN uses $\mathbf{f}_i$ for the detection computations.

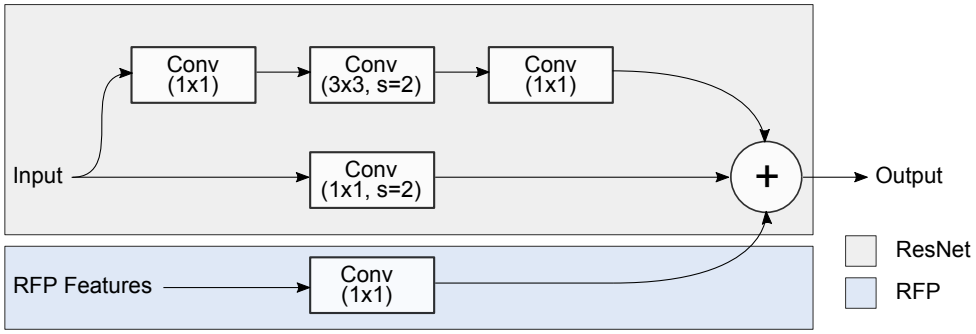



**Figure 6-3.** RFP adds transformed features to the first block of each stage of ResNet.

### 6.3.2 Recursive Feature Pyramid

Our proposed Recursive Feature Pyramid (RFP) adds feedback connections to FPN as highlighted in Fig. 6-2b. Let $\mathbf{R}_i$ denote the feature transformations before connecting them back to the bottom-up backbone. Then, $\forall i = 1, ..., S$, the output feature $\mathbf{f_i}$ of RFP is defined by

$$\mathbf{f}_i = \mathbf{F}_i\big(\mathbf{f}_{i+1}, \mathbf{x}_i\big), \quad \mathbf{x}_i = \mathbf{B}_i\big(\mathbf{x}_{i-1}, \mathbf{R}_i(\mathbf{f}_i)\big), \tag{6.2}$$

which makes RFP a recursive operation. We unroll it to a sequential network, *i.e.*, $\forall i = 1, ..., S, t = 1, ...T$,

$$\mathbf{f}_i^t = \mathbf{F}_i^t\big(\mathbf{f}_{i+1}^t, \mathbf{x}_i^t\big), \ \mathbf{x}_i^t = \mathbf{B}_i^t\big(\mathbf{x}_{i-1}^t, \mathbf{R}_i^t(\mathbf{f}_i^{t-1})\big), \tag{6.3}$$

where $T$ is the number of unrolled iterations, and we use superscript $t$ to denote operations and features at the unrolled step $t$. $\mathbf{f}_i^0$ is set to $\mathbf{0}$. In our implementation, $\mathbf{F}_i^t$ and $\mathbf{R}_i^t$ are shared across different steps. We show both shared and different $\mathbf{B}_i^t$ in the ablation study in Sec. 6.5 as well as the performances with different $T$'s. In our experiments, we use different $\mathbf{B}_i^t$ and set $T = 2$, unless otherwise stated.

We make changes to the ResNet [108] backbone $\mathbf{B}$ to allow it to take both $\mathbf{x}$ and $\mathbf{R}(\mathbf{f})$ as its input. ResNet has four stages, each of which is composed of several similar blocks. We only make changes to the first block of each stage, as shown in Fig. 6-3. This block computes a 3-layer feature and adds it to a feature computed by a shortcut. To use the feature $\mathbf{R}(\mathbf{f})$, we

add another convolutional layer with the kernel size set to $1$. The weight of this layer is initialized with $\mathbf{0}$ to make sure it does not have any real effect when we load the weights from a pretrained checkpoint.

### 6.3.3 ASPP as the Connecting Module

We use Atrous Spatial Pyramid Pooling (ASPP) [41] to implement the connecting module $\mathbf{R}$, which takes a feature $\mathbf{f}_i^t$ as its input and transforms it to the RFP feature used in Fig. 6-3. In this module, there are four parallel branches that take $\mathbf{f}_i^t$ as their inputs, the outputs of which are then concatenated together along the channel dimension to form the final output of $\mathbf{R}$. Three branches of them use a convolutional layer followed by a ReLU layer, the number of the output channels is $1/4$ the number of the input channels. The last branch uses a global average pooling layer to compress the feature, followed by a 1x1 convolutional layer and a ReLU layer to transform the compressed feature into a $1/4$-size (channel-wise) feature. Finally, it is resized and concatenated with the features from the other three branches. The convolutional layers in those three branches are of the following configurations: kernel size = [1, 3, 3], atrous rate = [1, 3, 6], padding = [0, 3, 6]. Unlike the original ASPP [41], we do not have a convolutional layer following the concatenated features as in here $\mathbf{R}$ does not generate the final output used in dense prediction tasks. Note that each of the four branches yields a feature with channels $1/4$ that of the input feature, and concatenating them generates a feature that has the same size as the input feature of $\mathbf{R}$. In Sec. 6.5, we show the performances of RFP with and without ASPP module.

### 6.3.4 Output Update by the Fusion Module

As shown in Fig. 6-2c, our RFP additionally uses a fusion module to combine $\mathbf{f}_i^t$ and $\mathbf{f}_i^{t+1}$ to update the values of $\mathbf{f}_i$ at the unrolled stage $t+1$ used in Eq. (6.3). The fusion module is very similar to the update process in recurrent neural networks [114] if we consider $\mathbf{f}_i^t$ as a sequence of data. The fusion module is used for unrolled steps from 2 to $T$. At the unrolled

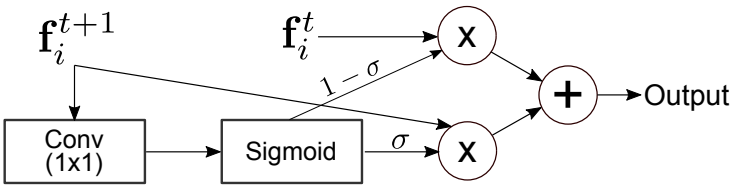



**Figure 6-4.** The fusion module used in RFP. $\sigma$ is the output of Sigmoid, which is used to fuse features from different steps.

step $t+1$ ($t = 1, ..., T-1$), the fusion module takes the feature $\mathbf{f}_i^t$ at the step $t$ and the feature $\mathbf{f}_i^{t+1}$ newly computed by FPN at the step $t+1$ as its input. The fusion module uses the feature $\mathbf{f}_i^{t+1}$ to compute an attention map by a convolutional layer followed by a Sigmoid operation. The resulting attention map is used to compute the weighted sum of $\mathbf{f}_i^t$ and $\mathbf{f}_i^{t+1}$ to form an updated $\mathbf{f}_i$. This $\mathbf{f}_i$ will be used as $\mathbf{f}_i^{t+1}$ for the computation in the following steps. In the ablation study in Sec. 6.5, we will show the performances of RFP with and without the fusion module.

## 6.4 Switchable Atrous Convolution

### 6.4.1 Atrous Convolution

Atrous convolution [42], [115], [205] is an effective technique to enlarge the field-of-view of filters at any convolutional layer. In particular, atrous convolution with atrous rate $r$ introduces $r-1$ zeros between consecutive filter values, equivalently enlarging the kernel size of a $k \times k$ filter to $k_e = k + (k-1)(r-1)$ without increasing the number of parameters or the amount of computation. Fig. 6-1b shows an example of a 3x3 convolutional layer with the atrous rate set to 1 (red) and 2 (green): the same kind of object of different scales could be roughly detected by the same set of convolutional weights using different atrous rates.

### 6.4.2 Switchable Atrous Convolution

In this subsection, we present the details of our proposed Switchable Atrous Convolution (SAC). Fig. 6-5 shows the overall architecture of SAC, which has three major components: two global context modules appended *before* and *after* the SAC component. This subsection focuses

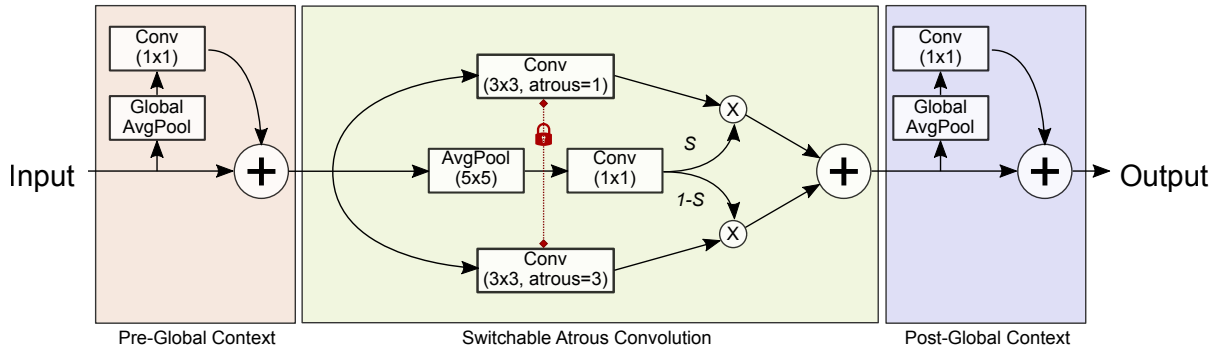



**Figure 6-5.** Switchable Atrous Convolution (SAC). We convert every 3x3 convolutional layer in the backbone ResNet to SAC, which softly switches the convolutional computation between different atrous rates. The **lock** indicates that the weights are the same except for a trainable difference (see Eq. 6.4). Two global context modules add image-level information to the features.

on the main SAC component in the middle and we will explain the global context modules afterward.

We use $\mathbf{y} = \textbf{Conv}(\mathbf{x}, \mathbf{w}, r)$ to denote the convolutional operation with weight $\mathbf{w}$ and atrous rate $r$ which takes $\mathbf{x}$ as its input and outputs $\mathbf{y}$. Then, we can convert a convolutional layer to SAC as follows.

$$\begin{aligned}\textbf{Conv}\big(\mathbf{x}, \mathbf{w}, 1\big) \xrightarrow[\text{to SAC}]{\text{Convert}} \mathbf{S}\big(\mathbf{x}\big) \cdot \textbf{Conv}\big(\mathbf{x}, \mathbf{w}, 1\big) \\ +\big(1 - \mathbf{S}(\mathbf{x})\big) \cdot \textbf{Conv}\big(\mathbf{x}, \mathbf{w} + \Delta\mathbf{w}, r\big)\end{aligned} \tag{6.4}$$

where $r$ here is a hyper-parameter of SAC, $\Delta\mathbf{w}$ is a trainable weight, and the switch function $\mathbf{S}(\cdot)$ is implemented as an average pooling layer with a 5x5 kernel followed by a 1x1 convolutional layer (see Fig. 6-5). The switch function is input and location dependent; thus, the backbone model is able to adapt to different scales as needed. We set $r = 3$ in our experiments, unless stated otherwise.

We propose a locking mechanism by setting one weight as $\mathbf{w}$ and the other as $\mathbf{w} + \Delta\mathbf{w}$ for the following reasons. Object detectors usually use pretrained checkpoints to initialize the weights. However, for a SAC layer converted from a standard convolutional layer, the weight for the larger atrous rate is missing. Since objects at different scales can be roughly detected by the same weight with different atrous rates, it is natural to initialize the missing weights with those in the pretrained model. Our implementation uses $\mathbf{w} + \Delta\mathbf{w}$ for the missing weight where

$\mathbf{w}$ is from the pretrained checkpoint and $\Delta\mathbf{w}$ is initialized with $\mathbf{0}$. When fixing $\Delta\mathbf{w} = 0$, we observe a drop of $0.1\%$ AP. But $\Delta\mathbf{w}$ alone without the locking mechanism degrades AP a lot.

### 6.4.3 Global Context

As shown in Fig. 6-5, we insert two global context modules before and after the main component of SAC. These two modules are light-weighted as the input features are first compressed by a global average pooling layer. The global context modules are similar to SENet [118] except for two major differences: (1) we only have one convolutional layer without any non-linearity layers, and (2) the output is added back to the main stream instead of multiplying the input by a re-calibrating value computed by Sigmoid. Experimentally, we found that adding the global context information before the SAC component (*i.e.*, adding global information to the switch function) has a positive effect on the detection performance. We speculate that this is because $\mathrm{S}$ can make more stable switching predictions when global information is available. We then move the global information outside the switch function and place it before and after the major body so that both **Conv** and $\mathrm{S}$ can benefit from it. We did not adopt the original SENet formulation as we found no improvement on the final model AP. In the ablation study in Sec. 6.5, we show the performances of SAC with and without the global context modules.

### 6.4.4 Implementation Details

In our implementation, we use deformable convolution [56], [329] to replace both of the convolutional operations in Eq. 6.4. The offset functions of them are not shared, which are initialized to predict $\mathbf{0}$ when loading from a pretrained backbone. Experiments in Sec. 6.5 will show performance comparisons of SAC with and without deformable convolution. We adopt SAC on ResNet and its variants [108], [281] by replacing all the 3x3 convolutional layers in the backbone. The weights and the biases in the global context modules are initialized with $\mathbf{0}$. The weight in the switch $\mathrm{S}$ is initialized with $\mathbf{0}$ and the bias is set to $\mathbf{1}$. $\Delta\mathbf{w}$ is initialized with $\mathbf{0}$. The above initialization strategy guarantees that when loading the backbone pretrained on

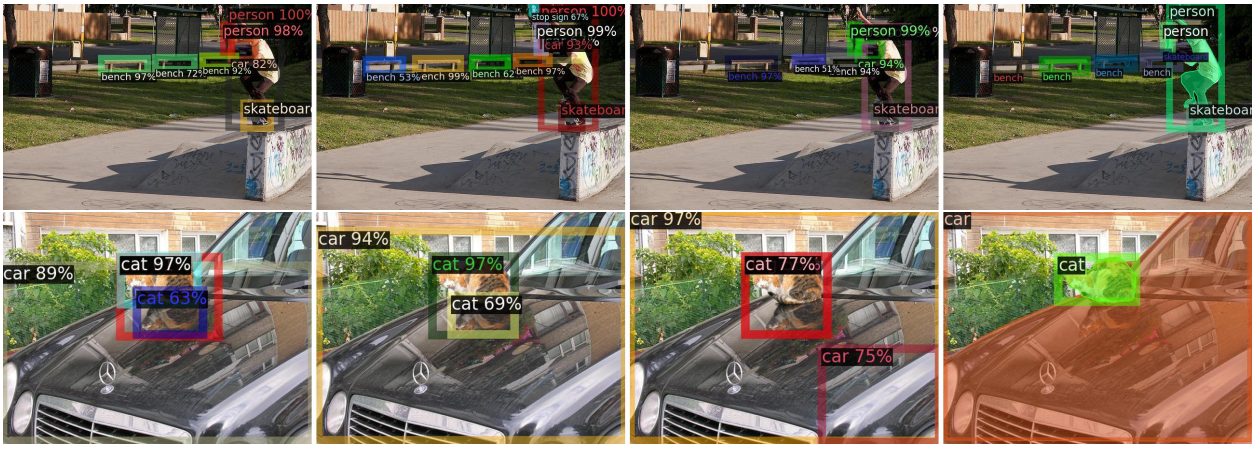



**Figure 6-6.** From left to right: visualization of the detection results by HTC, 'HTC + RFP', 'HTC + SAC' and the ground truth.

| HTC | RFP | SAC | Box AP | Box $AP_{50}$ | Box $AP_{75}$ | Box $AP_S$ | Box $AP_M$ | Box $AP_L$ | Mask AP | Mask $AP_{50}$ | Mask $AP_{75}$ | Mask $AP_S$ | Mask $AP_M$ | Mask $AP_L$ | Runtime FPS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ | | | 42.0 | 60.8 | 45.5 | 23.7 | 45.5 | 56.4 | 37.1 | 58.2 | 39.9 | 19.1 | 40.2 | 51.9 | 4.3 |
| ✓ | ✓ | | 46.2 | 65.1 | 50.2 | 27.9 | 50.3 | 60.3 | 40.4 | 62.5 | 43.5 | 22.3 | 43.8 | 54.9 | 4.1 |
| ✓ | | ✓ | 46.3 | 65.8 | 50.2 | 27.8 | 50.6 | 62.4 | 40.4 | 63.1 | 43.4 | 22.7 | 44.2 | 56.4 | 4.2 |
| ✓ | ✓ | ✓ | 49.0 | 67.7 | 53.0 | 30.1 | 52.6 | 64.9 | 42.1 | 64.8 | 45.5 | 23.9 | 45.6 | 57.8 | 3.9 |

**Table 6-II.** Detection results on COCO `val2017` with ResNet-50 as backbone. The models are trained for 12 epochs.

ImageNet [230], converting all the 3x3 convolutional layers to SAC will not change the output before taking any steps of training on COCO [173].

## 6.5 Experiments

### 6.5.1 Experimental Details

We conduct experiments on COCO dataset [173]. All the models presented in the chapter are trained on the split of `train2017` which has 115k labeled images. Then, we test the models on `val2017` and `test-dev`. We implement DetectoRS with mmdetection [37]. Our baseline model is HTC [36], which uses the bounding box and instance segmentation annotations from the dataset. Runtime is measured on a single NVIDIA TITAN RTX graphics card. We strictly follow the experimental settings of HTC [36]. For ablation studies, we train models for 12 epochs

| | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|
| Baseline HTC | 42.0 | 60.8 | 45.5 | 23.7 | 45.5 | 56.4 |
| RFP | 46.2 | 65.1 | 50.2 | 27.9 | 50.3 | 60.3 |
| RFP + sharing | 45.4 | 64.1 | 49.4 | 26.5 | 49.0 | 60.0 |
| RFP - aspp | 45.7 | 64.2 | 49.6 | 26.7 | 49.3 | 60.5 |
| RFP - fusion | 45.9 | 64.7 | 50.0 | 27.0 | 50.1 | 60.1 |
| RFP + 3X | 47.5 | 66.3 | 51.8 | 29.0 | 51.6 | 61.9 |
| SAC | 46.3 | 65.8 | 50.2 | 27.8 | 50.6 | 62.4 |
| SAC - DCN | 45.3 | 65.0 | 49.3 | 27.5 | 48.7 | 60.6 |
| SAC - DCN - global | 44.3 | 63.7 | 48.2 | 25.7 | 48.0 | 59.6 |
| SAC - DCN - locking | 44.7 | 64.4 | 48.7 | 26.0 | 48.7 | 59.0 |
| SAC - DCN + DS | 45.1 | 64.6 | 49.0 | 26.3 | 49.3 | 60.1 |

**Table 6-III.** Ablation study of RFP (the middle group) and SAC (the bottom group) on COCO `val2017` with ResNet-50.

with the learning rate multiplied by 0.1 after 8 and 12 epochs. Additionally, other training and testing settings are kept the same and no bells and whistles are used for them. For our main results after the ablation studies, we use multi-scale training with the long edge set to 1333 and the short edge randomly sampled from [400, 1200]. We train the models for 40 epochs with the learning rate multiplied by 0.1 after 36 and 39 epochs. Soft-NMS [24] is used for ResNeXt-101-32x4d and ResNeXt-101-64x4d. We also report the results with and without test-time augmentation (TTA), which includes horizontal flip and multi-scale testing with the short edge set to [800, 1000, 1200, 1400, 1600] and the long edge set to 1.5x short edge.

### 6.5.2 Ablation Studies

In this subsection, we show the ablation studies of RFP and SAC in Tab. 6-II and Tab. 6-III. Tab. 6-II shows the box and mask AP of the baseline HTC with ResNet-50 and FPN as its backbone. Then, we add our proposed RFP and SAC to the baseline HTC, both of which are able to improve AP by $> 4$% without too much decrease in the speed. Combining them together results in our DetectoRS which achieves $49$% box AP and 42.1% mask AP at 3.9 fps.

Tab. 6-III shows the individual ablation study of RFP and SAC where we present the sources of their improvements. For RFP, we show 'RFP + sharing' where $B_i^1$ and $B_i^2$ share their

| Method | Backbone | TTA | $AP_{bbox}$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|
| YOLOv3 [227] | DarkNet-53 | | 33.0 | 57.9 | 34.4 | 18.3 | 25.4 | 41.9 |
| RetinaNet [172] | ResNeXt-101 | | 40.8 | 61.1 | 44.1 | 24.1 | 44.2 | 51.2 |
| RefineDet [309] | ResNet-101 | ✓ | 41.8 | 62.9 | 45.7 | 25.6 | 45.1 | 54.1 |
| CornerNet [149] | Hourglass-104 | ✓ | 42.1 | 57.8 | 45.3 | 20.8 | 44.8 | 56.7 |
| ExtremeNet [321] | Hourglass-104 | ✓ | 43.7 | 60.5 | 47.0 | 24.1 | 46.9 | 57.6 |
| FSAF [326] | ResNeXt-101 | ✓ | 44.6 | 65.2 | 48.6 | 29.7 | 47.1 | 54.6 |
| FCOS [254] | ResNeXt-101 | | 44.7 | 64.1 | 48.4 | 27.6 | 47.5 | 55.6 |
| CenterNet [320] | Hourglass-104 | ✓ | 45.1 | 63.9 | 49.3 | 26.6 | 47.1 | 57.7 |
| NAS-FPN [88] | AmoebaNet | | 48.3 | - | - | - | - | - |
| SEPC [268] | ResNeXt-101 | | 50.1 | 69.8 | 54.3 | 31.3 | 53.3 | 63.7 |
| SpineNet [66] | SpineNet-190 | | 52.1 | 71.8 | 56.5 | 35.4 | 55.0 | 63.6 |
| EfficientDet-D7 [251] | EfficientNet-B6 | | 52.2 | 71.4 | 56.3 | - | - | - |
| EfficientDet-D7x (Model Zoo on GitHub) | - | - | 55.1 | 74.3 | 59.9 | 37.2 | 57.9 | 68.0 |
| Mask R-CNN [107] | ResNet-101 | | 39.8 | 62.3 | 43.4 | 22.1 | 43.2 | 51.2 |
| Cascade R-CNN [31] | ResNet-101 | | 42.8 | 62.1 | 46.3 | 23.7 | 45.5 | 55.2 |
| Libra R-CNN [204] | ResNeXt-101 | | 43.0 | 64.0 | 47.0 | 25.3 | 45.6 | 54.6 |
| DCN-v2 [329] | ResNet-101 | ✓ | 46.0 | 67.9 | 50.8 | 27.8 | 49.1 | 59.5 |
| PANet [177] | ResNeXt-101 | | 47.4 | 67.2 | 51.8 | 30.1 | 51.7 | 60.0 |
| SINPER [241] | ResNet-101 | ✓ | 47.6 | 68.5 | 53.4 | 30.9 | 50.6 | 60.7 |
| SNIP [240] | Model Ensemble | ✓ | 48.3 | 69.7 | 53.7 | 31.4 | 51.6 | 60.7 |
| TridentNet [166] | ResNet-101 | ✓ | 48.4 | 69.7 | 53.5 | 31.8 | 51.3 | 60.3 |
| Cascade Mask R-CNN [31] | ResNeXt-152 | ✓ | 50.2 | 68.2 | 54.9 | 31.9 | 52.9 | 63.5 |
| TSD [243] | SENet154 | ✓ | 51.2 | 71.9 | 56.0 | 33.8 | 54.8 | 64.2 |
| MegDet [206] | Model Ensemble | ✓ | 52.5 | - | - | - | - | - |
| CBNet [180] | ResNeXt-152 | ✓ | 53.3 | 71.9 | 58.5 | 35.5 | 55.8 | 66.7 |
| HTC [36] | ResNet-50 | | 43.6 | 62.6 | 47.4 | 24.8 | 46.0 | 55.9 |
| HTC | ResNeXt-101-32x4d | | 46.4 | 65.8 | 50.5 | 26.8 | 49.4 | 59.6 |
| HTC | ResNeXt-101-64x4d | | 47.2 | 66.5 | 51.4 | 27.7 | 50.1 | 60.3 |
| HTC + DCN [56] + multi-scale training | ResNeXt-101-64x4d | | 50.8 | 70.3 | 55.2 | 31.1 | 54.1 | 64.8 |
| DetectoRS | ResNet-50 | | 51.3 | 70.1 | 55.8 | 31.7 | 54.6 | 64.8 |
| DetectoRS | ResNet-50 | ✓ | 53.0 | 72.2 | 57.8 | 35.9 | 55.6 | 64.6 |
| DetectoRS | ResNeXt-101-32x4d | | 53.3 | 71.6 | 58.5 | 33.9 | 56.5 | 66.9 |
| DetectoRS | ResNeXt-101-32x4d | ✓ | 54.7 | 73.5 | 60.1 | 37.4 | 57.3 | 66.4 |
| DetectoRS | ResNeXt-101-64x4d | ✓ | 55.7 | 74.2 | 61.1 | 37.7 | 58.4 | 68.1 |

**Table 6-IV.** State-of-the-art comparison on COCO `test-dev` for bounding box object detection. TTA: test-time augmentation, which includes multi-scale testing, horizontal flipping, *etc*. The input size of DetectoRS without TTA is (1333, 800).

weights. We also demonstrate the improvements of the ASPP module and the fusion module by presenting the performance of RFP without them as in 'RFP - aspp' and 'RFP - fusion'. Finally, we increase the unrolled step $T$ from 2 to 3 and get 'RFP + 3X', which further improves the box AP by $1.3\%$. For SAC, we first experiment with SAC without DCN [56] (*i.e.*, 'SAC - DCN'). Then, we show that the global context is able to bring improvements on AP in 'SAC - DCN - global'. 'SAC - DCN - locking' breaks the locking mechanism in Fig. 6-5 where the second convolution uses only $\Delta \mathbf{w}$, proving that weight locking is necessary for SAC. Finally, in 'SAC - DCN + DS (dual-switch)', we replace $\mathbf{S}(x)$ and $1 - \mathbf{S}(x)$ with two independent switches $\mathbf{S}_1(x)$ and $\mathbf{S}_2(x)$. The ablation study in Tab. 6-III shows that the formulations of RFP and SAC have the best configuration within the design space we have explored.

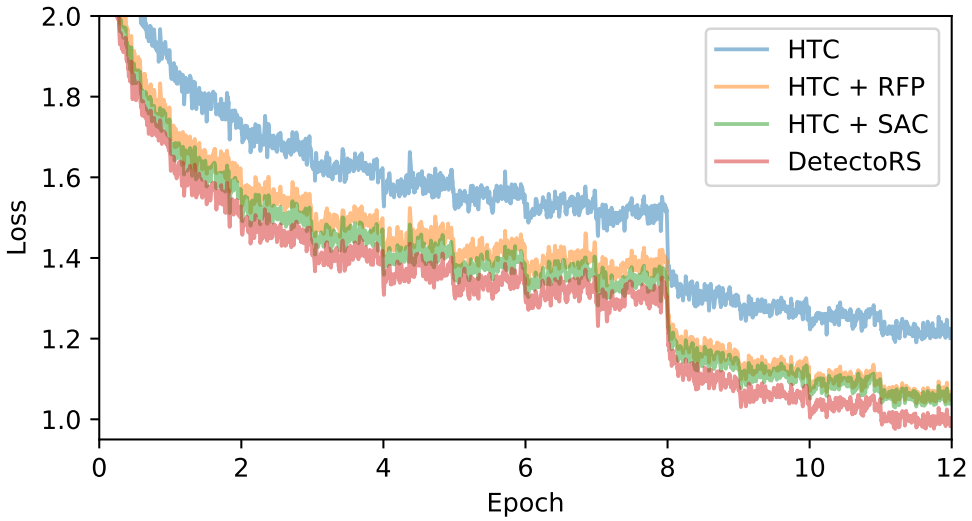



**Figure 6-7.** Comparing training losses of HTC, 'HTC + RFP', 'HTC + SAC', and DetectoRS during 12 training epochs.

Fig. 6-6 provides visualization of the results by HTC, 'HTC + RFP' and 'HTC + SAC'. From this comparison, we notice that RFP, similar to the human visual perception that selectively enhances or suppresses neuron activations, is able to find occluded objects more easily for which the nearby context information is more critical. SAC, because of its ability to increase the field-of-view as needed, is more capable of detecting large objects in the images. This is also consistent with the results of SAC shown in Tab. 6-II where it has a higher $AP_L$. Fig. 6-7 shows the training losses of HTC, 'HTC + RFP', 'HTC + SAC', and DetectoRS. Both are able to significantly accelerate the training process and converge to lower losses.

### 6.5.3 Main Results

In this subsection, we show the main results of DetectoRS. We equip the state-of-art detector HTC with DetectoRS, and use ResNet-50 and ResNeXt-101 as the backbones for DetectoRS. The bounding box detection results are shown in Tab. 6-IV. The results are divided into 4 groups. The first group shows one-stage detectors. The second group shows multi-stage detectors. The third group is HTC, which is the baseline of DetectoRS. The fourth group is our results. The results can be also categorized as simple test results and TTA results, where

**Figure 6-8.** Visualizing the outputs of the learned switch functions in Switchable Atrous Convolution. Darker intensity means that the switch function for that region gathers more outputs from the larger atrous rate.

| Method | Backbone | TTA | $AP_{mask}$ | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|
| HTC [36] | ResNet-50 | | 38.5 | 60.1 | 41.7 | 20.4 | 40.6 | 51.2 |
| HTC | ResNeXt-101-32x4d | | 40.7 | 63.2 | 44.1 | 22.0 | 43.3 | 54.2 |
| HTC | ResNeXt-101-64x4d | | 41.3 | 63.9 | 44.8 | 22.7 | 44.0 | 54.7 |
| HTC + DCN [56] + multi-scale training | ResNeXt-101-64x4d | | 44.2 | 67.8 | 48.1 | 25.3 | 47.2 | 58.7 |
| DetectoRS | ResNet-50 | | 44.4 | 67.7 | 48.3 | 25.6 | 47.5 | 58.3 |
| DetectoRS | ResNet-50 | ✓ | 45.8 | 69.8 | 50.1 | 29.2 | 48.3 | 58.2 |
| DetectoRS | ResNeXt-101-32x4d | | 45.8 | 69.2 | 50.1 | 27.4 | 48.7 | 59.6 |
| DetectoRS | ResNeXt-101-32x4d | ✓ | 47.1 | 71.1 | 51.6 | 30.3 | 49.5 | 59.6 |
| DetectoRS | ResNeXt-101-64x4d | ✓ | 48.5 | 72.0 | 53.3 | 31.6 | 50.9 | 61.5 |

**Table 6-V.** Instance segmentation comparison on COCO `test-dev`.

TTA is short for test-time augmentation. The third column shows whether TTA is used. Note that different methods use different TTA strategies. For example, CBNet uses a strong TTA strategy, which can improve their box AP from 50.7% to 53.3%. Our TTA strategy only brings 1.4% improvement when using ResNeXt-101-32x4d as backbone. The simple test settings can also vary significantly among different detectors. DetectoRS uses (1333, 800) as the test image size. Larger input sizes tend to bring improvements (see [251]). DetectoRS adopts the same setting of HTC.

We also show the instance segmentation results in Tab. 6-V. As many methods in Tab. 6-IV do not provide mask AP in their paper, we only compare DetectoRS with its baseline HTC. The experimental settings for bounding box and mask object detection are the same except that we report $AP_{mask}$ instead of $AP_{bbox}$. From Tab. 6-V, we can see that consistent with the

| Method | TTA | PQ | $PQ^{Th}$ | $PQ^{St}$ |
|---|---|---|---|---|
| DeeperLab [298] | | 34.3 | 37.5 | 29.6 |
| SSAP [83] | ✓ | 36.9 | 40.1 | 32.0 |
| Panoptic-DeepLab [52] | ✓ | 41.4 | 45.1 | 35.9 |
| Axial-DeepLab-L [264] | ✓ | 44.2 | 49.2 | 36.8 |
| TASCNet [163] | | 40.7 | 47.0 | 31.0 |
| Panoptic-FPN [135] | | 40.9 | 48.3 | 29.7 |
| AdaptIS [242] | ✓ | 42.8 | 53.2 | 36.7 |
| AUNet [167] | | 46.5 | 55.8 | 32.5 |
| UPSNet [283] | ✓ | 46.6 | 53.2 | 36.7 |
| Li *et al.* [164] | | 47.2 | 53.5 | 37.7 |
| SpatialFlow [45] | ✓ | 47.3 | 53.5 | 37.9 |
| SOGNet [299] | ✓ | 47.8 | - | - |
| DetectoRS | ✓ | 50.0 | 58.5 | 37.2 |

**Table 6-VI.** State-of-the-art comparison on COCO `test-dev` for panoptic segmentation.

bounding box results, DetectoRS also brings significant improvements over its baseline for instance segmentation.

Finally, the panoptic segmentation results are presented in Tab. 6-VI. As DetectoRS only detects things, we use the stuff predictions by DeepLabv3+ [44] with backbone Wide-ResNet-41 [40], [277], [305]. Combing the thing and the stuff predictions using the script available in panoptic API [136] without tuning any hyper-parameters, we set a new state-of-the-art of $50.0\%$ PQ for panoptic segmentation on COCO.

### 6.5.4 Visualizing Learned Switches

Fig. 6-8 shows the visualization results of the outputs of the last switch function of 'SAC - DCN' in Tab. 6-III. Darker intensity in the figure means that the switch function for that region gathers more outputs from the larger atrous rate. Comparing the switch outputs with the original images, we observe that the switch outputs are well aligned with the ground-truth object scales. These results prove that the behaviors of Switchable Atrous Convolution are consistent with our intuition, which tend to use larger atrous rates when encountering large objects.

## 6.6 Conclusion

In this chapter, motivated by the design philosophy of looking and thinking twice, we have proposed DetectoRS, which includes Recursive Feature Pyramid and Switchable Atrous Convolution. Recursive Feature Pyramid implements thinking twice at the macro level, where the outputs of FPN are brought back to each stage of the bottom-up backbone through feedback connections. Switchable Atrous Convolution instantiates looking twice at the micro level, where the inputs are convolved with two different atrous rates. DetectoRS is tested on COCO for object detection, instance segmentation, and panoptic segmentation. It sets new state-of-the-art results on all these tasks.

# Chapter 7

# ViP-DeepLab: Learning Visual Perception with Depth-aware Video Panoptic Segmentation

In this chapter, we present ViP-DeepLab, a unified model attempting to tackle the long-standing and challenging inverse projection problem in vision, which we model as restoring the point clouds from perspective image sequences while providing each point with instance-level semantic interpretations. Solving this problem requires the vision models to predict the spatial location, semantic class, and temporally consistent instance label for each 3D point. ViP-DeepLab approaches it by jointly performing monocular depth estimation and video panoptic segmentation. We name this joint task as Depth-aware Video Panoptic Segmentation, and propose a new evaluation metric along with two derived datasets for it, which will be made available to the public. On the individual sub-tasks, ViP-DeepLab also achieves state-of-the-art results, outperforming previous methods by 5.1% VPQ on Cityscapes-VPS, ranking 1st on the KITTI monocular depth estimation benchmark, and 1st on KITTI MOTS pedestrian.

## 7.1 Introduction

The inverse projection problem, one of the most fundamental problems in vision, refers to the ambiguous mapping from the retinal images to the sources of retinal stimulation. Such a mapping requires retrieving all the visual information about the 3D environment using the

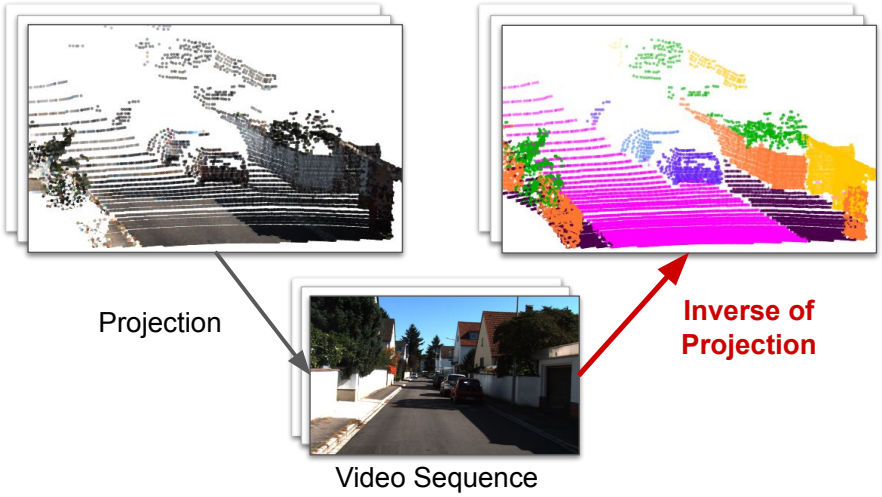



**Figure 7-1.** Projecting 3D points to the image plane results in 2D images. We study the inverse projection problem: how to restore the 3D points from 2D image sequences while providing temporally consistent instance-level semantic interpretations for the 3D points.

limited signals contained in the 2D images [202], [210]. Humans are able to easily establish this mapping by identifying objects, determining their sizes, and reconstructing the 3D scene layout, *etc*. To endow machines with similar abilities to visually perceive the 3D world, we aim to develop a model to tackle the inverse projection problem.

As a step towards solving the inverse projection, the problem is simplified as restoring the 3D point clouds with semantic understandings from the perspective image sequences, which calls for vision models to predict the spatial location, semantic class, and temporally consistent instance label for each 3D point. Fig. 7-1 shows an example of the inverse projection problem we study in this chapter. This simplified problem can be formulated as Depth-aware Video Panoptic Segmentation (DVPS) that contains two sub-tasks: (i) monocular depth estimation [234], which is used to estimate the spatial position of each 3D point that is projected to the image plane, and (ii) video panoptic segmentation [132], which associates the 3D points with temporally consistent instance-level semantic predictions.

For the new task DVPS, we present two derived datasets accompanied by a new evaluation metric named Depth-aware Video Panoptic Quality (DVPQ). DVPS datasets are hard to collect, as they need special depth sensors and a huge amount of labeling efforts. Existing

datasets usually lack some annotations or are not in the format for DVPS. Our solution is to augment and convert existing datasets for DVPS, producing two new datasets, Cityscapes-DVPS and SemKITTI-DVPS. Cityscapes-DVPS is derived from Cityscapes-VPS [132] by adding depth annotations from Cityscapes dataset [53], while SemKITTI-DVPS is derived from SemanticKITTI [19] by projecting its annotated 3D point clouds to the image plane. Additionally, the proposed metric DVPQ includes the metrics for depth estimation and video panoptic segmentation, requiring a vision model to simultaneously tackle the two sub-tasks. To this end, we present ViP-DeepLab, a unified model that jointly performs video panoptic segmentation and monocular depth estimation for each pixel on the image plane. In the following, we introduce how ViP-DeepLab tackles the two sub-tasks.

The first sub-task of DVPS is video panoptic segmentation [132]. Panoptic segmentation [136] unifies semantic segmentation [110] and instance segmentation [105] by assigning every pixel a semantic label and an instance ID. It has been recently extended to the video domain, resulting in video panoptic segmentation [132], which further demands each instance to have the same instance ID throughout the video sequence. This poses additional challenges to panoptic segmentation as the model is now expected to be able to track objects in addition to detecting and segmenting them. Current approach VPSNet [132] adds a tracking head to learn the correspondence between the instances from different frames based on their regional feature similarity. By contrast, our ViP-DeepLab takes a different approach to tracking objects. Specifically, motivated by our finding that video panoptic segmentation can be modeled as concatenated image panoptic segmentation, we extend Panoptic-DeepLab [52] to perform center regression for *two* consecutive frames with respect to *only* the object centers that appear in the *first* frame. During inference, this offset prediction allows ViP-DeepLab to group all the pixels in the two frames to the same object that appears in the first frame. New instances emerge if they are not grouped to the previously detected instances. This inference process continues for every two consecutive frames (with one overlapping frame) in a video sequence, stitching panoptic predictions together to form predictions with temporally consistent instance IDs. Based

on this simple design, our ViP-DeepLab outperforms VPSNet [132] by a large margin of 5.1% VPQ, setting a new record on the Cityscapes-VPS dataset [132]. Additionally, Multi-Object Tracking and Segmentation (MOTS) [261] is a similar task to video panoptic segmentation, but only segments and tracks two classes: pedestrians and cars. We therefore also apply our ViP-DeepLab to MOTS. As a result, ViP-DeepLab outperforms the current state-of-the-art PointTrack [292] by 7.2% and 2.5% sMOTSA on pedestrians and cars, respectively, and ranks 1st on the leaderboard for KITTI MOTS pedestrian.

The second sub-task of DVPS is monocular depth estimation, which is challenging for both computers [234] and humans [117]. The state-of-the-art methods are mostly based on deep networks trained in a fully-supervised way [62], [67], [68], [81]. Following the same direction, our ViP-DeepLab appends another depth prediction head on top of Panoptic-DeepLab [52]. Without using any additional *depth* training data, such a simple approach outperforms all the published and unpublished works on the KITTI benchmark [87]. Specifically, it outperforms DORN [81] by 0.97 SILog, and even outperforms MPSD that uses extra planet-scale depth data [5], breaking the long-standing record on the challenging KITTI depth estimation [257]. Notably, the differences between top-performing methods are all around 0.1 SILog, while our method significantly outperforms them.

To summarize, our contributions are listed as follows.

- We propose a new task Depth-aware Video Panoptic Segmentation (DVPS), as a step towards solving the inverse projection problem by formulating it as joint video panoptic segmentation [132] and monocular depth estimation [234].
- We present two DVPS datasets along with an evaluation metric Depth-aware Video Panoptic Quality (DVPQ). To facilitate future research, the datasets and the evaluation codes are made publicly available.
- We develop ViP-DeepLab, a unified model for DVPS. On the individual sub-tasks, ViP-DeepLab ranks 1st on Cityscapes-VPS [132], KITTI-MOTS pedestrian [261], and KITTI monocular depth estimation [87].

## 7.2 Related Work

**Panoptic Segmentation** Recent methods for image panoptic segmentation can be grouped into two types: top-down (proposal-based) methods and bottom-up (box-free) methods. Top-down methods employ a two-stage approach which generates object proposals followed by outputting panoptic predictions based on regional computations [47], [150], [163], [164], [167], [214], [242], [275], [283]. For example, Panoptic FPN [136] incorporates a semantic segmentation head into Mask R-CNN [107]. Porzi *et al*. [211] propose a novel segmentation head to integrate FPN [171] features by a lightweight DeepLab-like module [41]. Bottom-up panoptic segmentation methods group pixels to form instances on top of semantic segmentation prediction [263], [264], [298]. For example, SSAP [83] uses pixel-pair affinity pyramid [179] and a cascaded graph partition module [131] to generate instances from coarse to fine. BBFNet [25] uses Hough-voting [17], [159] and Watershed transform [14], [259] to generate instance segmentation predictions. Panoptic-DeepLab [52] employs class-agnostic instance center regression [130], [199], [256] on top of semantic segmentation outputs from DeepLab [42], [44].

**Object Tracking** One of the major tasks in video panoptic segmentation is object tracking. Many trackers use tracking-by-detection, which divides the task into two sub-tasks where an object detector (*e.g.*[76], [228]) finds all objects and then an algorithm associates them [21], [73], [151], [212], [235], [237], [252], [274], [289], [327]. Another design is transforming object detectors to object trackers which detect and track objects at the same time [20], [75], [207], [271], [310], [312]. For example, CenterTrack [319] extends CenterNet [320] to predict offsets from the object center to its center in the previous frame. STEm-Seg [8] proposes to group all instance pixels in a video clip by learning a spatio-temporal embedding. By contrast, our ViP-DeepLab implicitly performs object tracking by clustering all instance pixels in two consecutive video frames. Additionally, our method simply uses center regression and achieves better results on MOTS [261].

**Monocular Depth Estimation** Monocular depth estimation predicts depth from a single image.

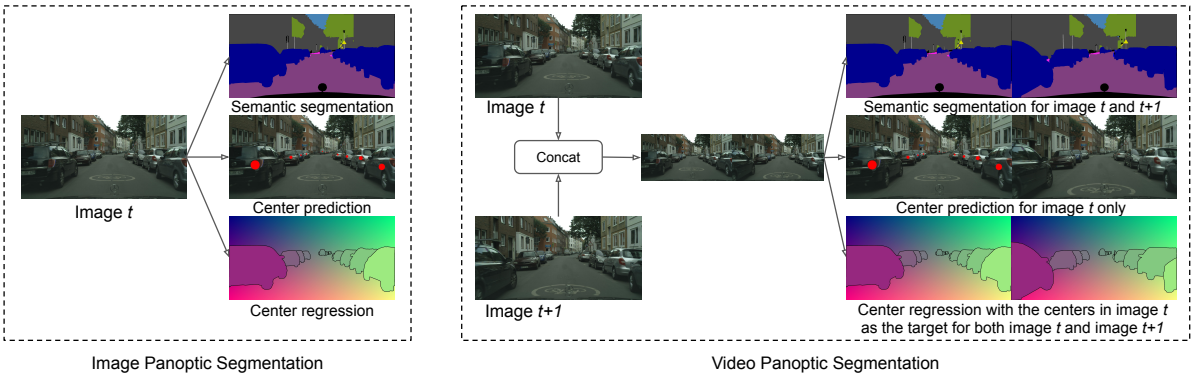



**Figure 7-2.** Comparing image panoptic segmentation and video panoptic segmentation. Our method is based on the finding that video panoptic segmentation can be modeled as concatenated image panoptic segmentation. Center regression is an offset map from each pixel to its object center. Here we draw the predicted centers instead of the offsets for clearer visualization.

It can be learned in a supervised way [34], [67], [81], [82], [157], [160], [234], [286], [301], by reconstructing images in the stereo setting [84], [91], [92], [143], [280], from videos [188], [262], [302], in relative order [46], or by joint learning with other vision tasks [258], [266], [285]. ViP-DeepLab models monocular depth estimation as a dense regression problem, and we train it in a fully-supervised manner.

## 7.3 ViP-DeepLab

In this section, we present ViP-DeepLab, which extends Panoptic-DeepLab [52] to jointly perform video panoptic segmentation [132] and monocular depth estimation [234].

### 7.3.1 Video Panoptic Segmentation

#### 7.3.1.1 Rethinking Image and Video Panoptic Segmentation

In the task of video panoptic segmentation, each instance is represented by a tube on the image plane and the time axis when the frames are stacked up. Given a clip $I^{t:t+k}$ with time window $k$, true positive (TP) is defined by $\text{TP} = \{(u, \hat{u}) \in U \times \hat{U} : \text{IoU}(u, \hat{u}) > 0.5\}$ where $U$ and $\hat{U}$ are the set of the ground-truth and predicted tubes, respectively. False positives (FP) and false negatives (FN) are defined accordingly. After accumulating the $\text{TP}_c$, $\text{FP}_c$, and $\text{FN}_c$

on all the clips with window size $k$ and class $c$, the evaluation metric Video Panoptic Quality (VPQ) [132] is defined by

$$\text{VPQ}^k = \frac{1}{\text{N}_{\text{classes}}} \sum_c \frac{\sum_{(u,\hat{u})\in \text{TP}_c} \text{IoU}(u,\hat{u})}{|\text{TP}_c| + \frac{1}{2}|\text{FP}_c| + \frac{1}{2}|\text{FN}_c|} \tag{7.1}$$

PQ [136] is thus equal to $\text{VPQ}^1$ (*i.e.*, $k = 1$).

Our method is based on the connection between PQ and VPQ. For an image sequence $I_t$ ($t = 1, ..., T$), let $P_t$ denote the panoptic prediction and $Q_t$ be the ground-truth panoptic segmentation. As $\text{VPQ}^k$ accumulates the PQ-related statistics from $P_t$ and $Q_t$ within a window of size $k$, we have

$$\text{VPQ}^k\Big(\Big[P_t, Q_t\Big]_{t=1}^{T}\Big) = \text{PQ}\Big(\Big[\,\|_{i=t}^{t+k-1} P_i, \|_{i=t}^{t+k-1} Q_i\Big]_{t=1}^{T-k+1}\Big) \tag{7.2}$$

where $\|_{i=t}^{t+k-1} P_i$ denotes the horizontal concatenation of $P_i$ from $t$ to $t+k-1$, and $\big[P_t, Q_t\big]_{t=1}^{T}$ denotes a list of pairs of $(P_t, Q_t)$ from $1$ to $T$ as the function input.

Eq. (7.2) reveals an interesting finding that video panoptic segmentation could be formulated as image panoptic segmentation with the images concatenated. Such a finding motivates us to extend image panoptic segmentation models to video panoptic segmentation with extra modifications.

#### 7.3.1.2 From Image to Video Panoptic Segmentation

Panoptic-DeepLab [52] approaches the problem of image panoptic segmentation by solving three sub-tasks: (1) semantic predictions for both 'thing' and 'stuff' classes, (2) center prediction for each instance of 'thing' classes, and (3) center regression for each pixel of objects. Fig. 7-2 shows an example of the tasks on the left. During inference, object centers with high confidence scores are kept, and each 'thing' pixel is associated with the closest object center to form object instances. Combining this 'thing' prediction and the 'stuff' prediction from semantic segmentation, Panoptic-DeepLab [52] generates the final panoptic prediction.

Our method extends Panoptic-DeepLab [52] to perform video panoptic segmentation. As the right part of Fig. 7-2 shows, it also breaks down the task of video panoptic segmentation

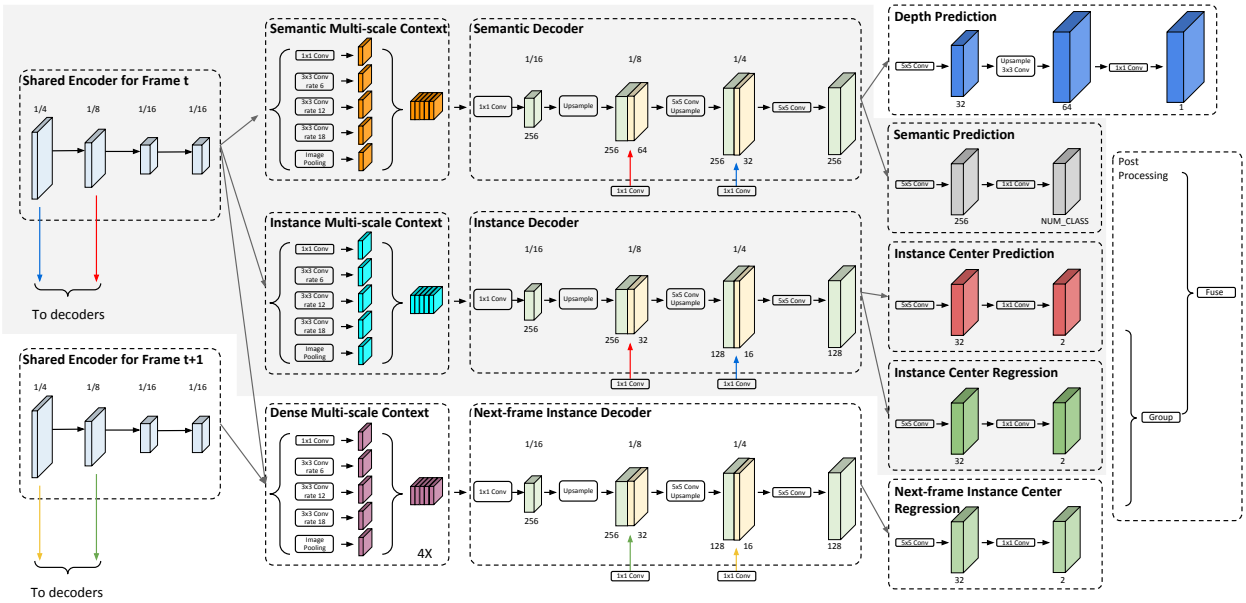



**Figure 7-3.** ViP-DeepLab extends Panoptic-DeepLab [52] (the gray part) by adding a depth prediction head to perform monocular depth estimation and a next-frame instance branch which regresses to the object centers in frame $t$ for frame $t+1$.

into three sub-tasks: semantic segmentation, center prediction, and center regression. During inference, our method takes image $t$ and $t+1$ concatenated horizontally as input, and only predicts the centers in image $t$. The center regression for both $t$ and $t+1$ will regress to the object centers in image $t$. By doing so, our method detects the objects in the first frame, and finds all the pixels belonging to them in the first and the second frames. Objects that appear only in the second frame are ignored here and will emerge again when the model works on the next image pair (*i.e.*, $(t+1, t+2)$). Our method models video panoptic segmentation as concatenated image panoptic segmentation, highly consistent with the definition of the metric VPQ.

Fig. 7-3 shows the architecture of our method. In order to perform the inference described above, we take image $t$ and $t+1$ as the input during training, and use the features of image $t$ to predict semantic segmentation, object centers, and center offsets for image $t$. In addition to that, we add a *next-frame instance branch* that predicts the center offsets for the pixels in image $t+1$ with respect to the centers in image $t$. The backbone features of image $t$ and $t+1$ are

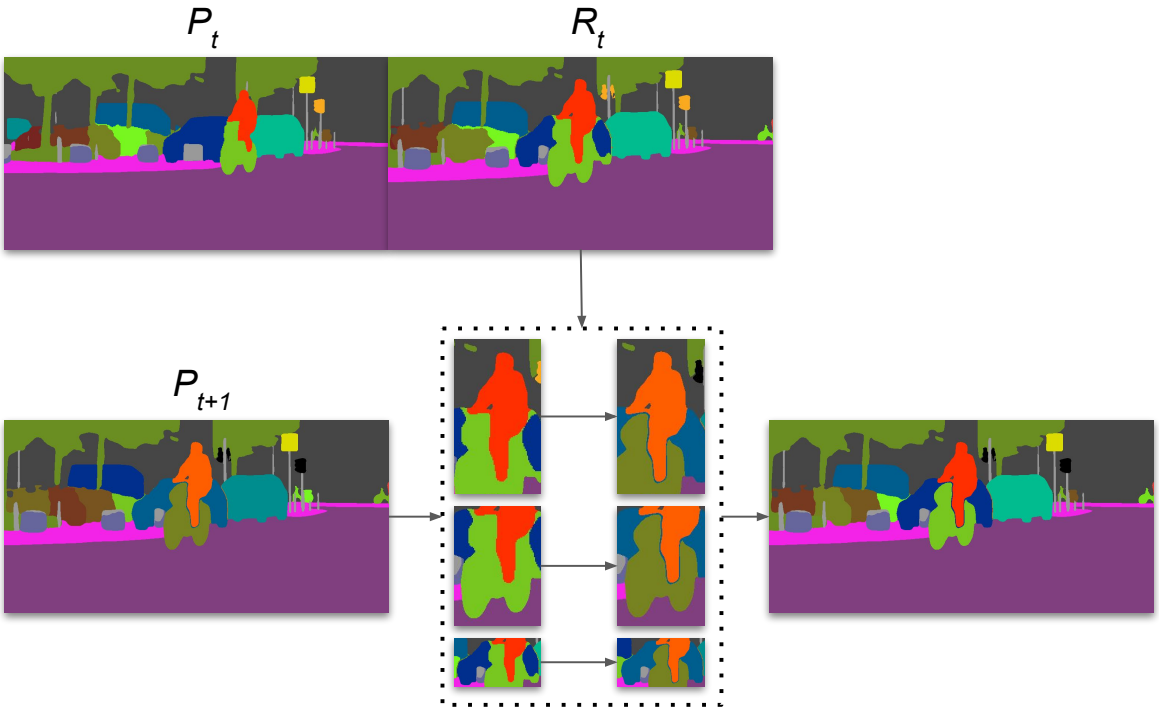



**Figure 7-4.** Visualization of stitching video panoptic predictions. It propagates IDs based on mask IoU between region pairs. ViP-DeepLab is capable of tracking objects with large movements, *e.g.*, the cyclist in the image. Panoptic prediction of $R_t$ is of high quality, which is why a simple IoU-based stitching method works well in practice.

concatenated along the feature axis before the *next-frame instance branch*. As their backbone features are separated before concatenation, the *next-frame instance branch* needs a large receptive field to perform long-range center regression. To address this, we use four ASPP modules in the branch, the output of which are densely-connected [120], [296] to dramatically increase the receptive field. We name this densely-connected module as Cascade-ASPP. Finally, the decoder in the *next-frame instance branch* uses the backbone features of image $t+1$ while the other branches use those of image $t$, as indicated by the colored arrows in the figure.

#### 7.3.1.3 Stitching Video Panoptic Predictions

Our method outputs panoptic predictions with temporally consistent IDs for two consecutive frames. To generate predictions for the entire sequence, we need to stitch the panoptic predictions. Fig 7-4 shows an example of our stitching method. For each image pair $t$ and $t+1$, we split the panoptic prediction of the concatenated input in the middle, and use $P_t$ to denote the left prediction, and $R_t$ to denote the right one. By doing so, $P_t$ becomes the panoptic prediction of image $t$, and $R_t$ becomes the panoptic prediction of image $t+1$ with instance IDs that are consistent with those of $P_t$. The goal of stitching is to propagate IDs from $R_t$ to $P_{t+1}$ so that each object in $P_t$ and $P_{t+1}$ will have the same ID.

The ID propagation is based on mask IoU between region pairs. For each region pair in $R_t$ and $P_{t+1}$, if they have the same class, and both find each other to have the largest mask IoU, then we propagate the ID between them. Objects that do not receive IDs will become new instances.

### 7.3.2 Monocular Depth Estimation

We model monocular depth estimation as a dense regression problem [68], where each pixel will have an estimated depth. As shown in Fig. 7-3, we add a depth prediction head on top of the decoded features of the semantic branch (*i.e.*, Semantic Decoder), which upsamples the

features by 2x and generates logits $f_d$ for depth regression:

$$\text{Depth} = \text{MaxDepth} \times \text{Sigmoid}(f_d) \tag{7.3}$$

MaxDepth controls the range of the predicted depth, which is set to $88$ for the range (about $0$ to $80$m) of KITTI [257].

Many metrics have been proposed to evaluate the quality of monocular depth prediction [87]. Among them, scale-invariant logarithmic error [68] and relative squared error [87] are popular ones, which could also be directly optimized as training loss functions. We therefore combine them to train our depth prediction. Specifically, let $d$ and $\hat{d}$ denote the ground-truth and the predicted depth, respectively. Our loss function for depth estimation is then defined by

$$\begin{aligned}\mathcal{L}_{\text{depth}}(d, \hat{d}) =& \frac{1}{n}\sum_i \Big(\log d_i - \log \hat{d}_i\Big)^2 - \frac{1}{n^2}\Big(\sum_i \log d_i \\ & - \log \hat{d}_i\Big)^2 + \Big(\frac{1}{n}\sum_i \big(\frac{d_i - \hat{d}_i}{d_i}\big)^2\Big)^{0.5}\end{aligned} \tag{7.4}$$

### 7.3.3 Depth-aware Video Panoptic Segmentation

Motivated by solving the inverse projection problem, we introduce a challenging task, Depth-aware Video Panoptic Segmentation (DVPS), unifying the problems of monocular depth estimation and video panoptic segmentation. In the task of DVPS, images are densely annotated with a tuple $(c, id, d)$ for each labeled pixel, where $c$, $id$ and $d$ denote its semantic class, instance ID and depth. The model is expected to also generate a tuple $(\hat{c}, \hat{id}, \hat{d})$ for each pixel.

To evaluate methods for DVPS, we propose a metric called Depth-aware Video Panoptic Quality (DVPQ), which extends VPQ by additionally considering the depth prediction with the inlier metric. Specifically, let $P$ and $Q$ be the prediction and ground-truth, respectively. We use $P_i^c$, $P_i^{id}$ and $P_i^d$ to denote the predictions of example $i$ on the semantic class, instance ID, and depth. The notations also apply to $Q$. Let $k$ be the window size (as in Eq. (7.2)) and $\lambda$ be the depth threshold. Then, $\text{DVPQ}_\lambda^k(P, Q)$ is defined by

$$\text{PQ}\Big(\Big[\ \|_{i=t}^{t+k-1}\ \big(\hat{P}_i^c, P_i^{id}\big), \|_{i=t}^{t+k-1}\big(Q_i^c, Q_i^{id}\big)\Big]_{t=1}^{T-k+1}\Big) \tag{7.5}$$

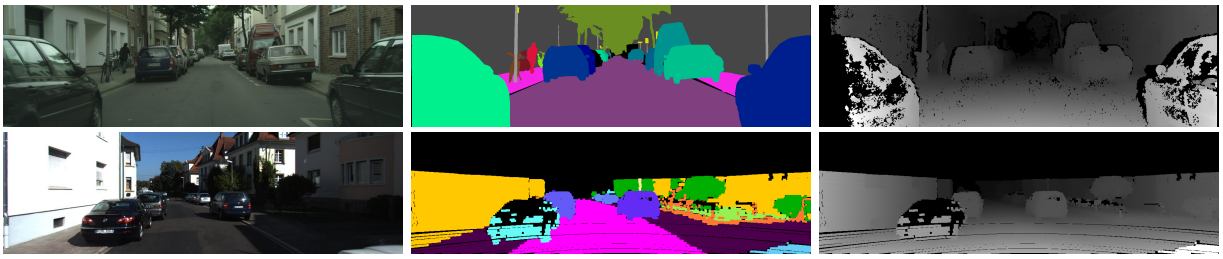


**Figure 7-5.** Dataset examples of Cityscapes-DVPS (top) and SemKITTI-DVPS (bottom). From left to right: input image, video panoptic segmentation annotation, and depth map. Regions are black if they are not covered by the Velodyne data or they are removed by the data preprocessing step including disparity consistency check and non-foreground suppression.

where $\hat{P}_i^c = P_i^c$ for pixels that have *absolute relative* depth errors under $\lambda$ (*i.e.*, $|P_i^d - Q_i^d| \leq \lambda Q_i^d$), and will be assigned a void label otherwise. In other words, $\hat{P}_i^c$ filters out pixels that have large absolute relative depth errors. As a result, the metrics VPQ [132] (also image PQ [136]) and depth inlier metric (*i.e.*, $\max(P_i^d/Q_i^d, Q_i^d/P_i^d) = \delta <$ threshold) [68] can be *approximately* viewed as special cases for DVPQ.

Following [132], we evaluate DVPQ$_\lambda^k$ for four different values of $k$ (depending on the dataset) and three values of $\lambda = \{0.1, 0.25, 0.5\}$. Those values of $\lambda$ approximately correspond to the depth inlier metric $\delta < 1.1$, $\delta < 1.25$, and $\delta < 1.5$, respectively. They are harder than the thresholds $1.25$, $1.25^2$ and $1.25^3$ that are commonly used in depth evaluation. We choose harder thresholds as many methods are able to get $> 99\%$ on the previous metrics [81], [157] . Larger $k$ and smaller $\lambda$ correspond to a higher accuracy requirement for a long-term consistency of joint video panoptic segmentation and depth estimation. The final number DVPQ is obtained by averaging over all values of $k$ and $\lambda$.

## 7.4 Datasets

To evaluate on the new task, Depth-aware Video Panoptic Segmentation, we create two new datasets, Cityscapes-DVPS and SemKITTI-DVPS. Fig. 7-5 shows two examples, one for each dataset. The details are elaborated below.

### 7.4.1 Cityscapes-DVPS

The original Cityscapes [53] only contains image-level panoptic annotations. Recently, Kim *et al*.introduce a video panoptic segmentation dataset Cityscapes-VPS [132] by further annotating 6 frames out of each 30-frame video sequence (with a gap of 5 frames between each annotation), resulting in totally 3,000 annotated frames where the training, validation, and test sets have 2,400, 300, and 300 frames, respectively. In the dataset, there are 19 semantic classes, including 8 'thing' and 11 'stuff' classes.

Even though Cityscapes-VPS contains video panoptic annotations, the depth annotations are missing. We find that the depth annotations could be converted from the disparity maps via stereo images, provided by the original Cityscapes dataset [53]. However, the quality of the pre-computed disparity maps is not satisfactory. To improve it, we select several modern disparity estimation methods [98], [113], [307], [311] and follow the process similar to [53]. Nevertheless, to discourage reproducing the depth generation process (so that one may game the benchmark), we do not disclose the details (*e.g.*, the exact employed disparity method). The depth annotations are made publicly available.

### 7.4.2 SemKITTI-DVPS

SemanticKITTI dataset [19] is based on the odometry dataset of the KITTI Vision benchmark [87]. The dataset splits 22 sequences in to 11 training sequences and 11 test sequences. The training sequence 08 is used for validation. This dataset includes 8 'thing' and 11 'stuff' classes.

SemanticKITTI dataset provides perspective images and panoptic-labeled 3D point clouds (*i.e.*, semantic class and instance ID are annotated). To convert it for our use, we project the 3D point clouds into the image plane. However, there are two challenges when converting the dataset, as presented in Fig. 7-6. The first problem is that some point clouds are not visible to the camera but are recorded and labeled. For example, the first row of Fig. 7-6

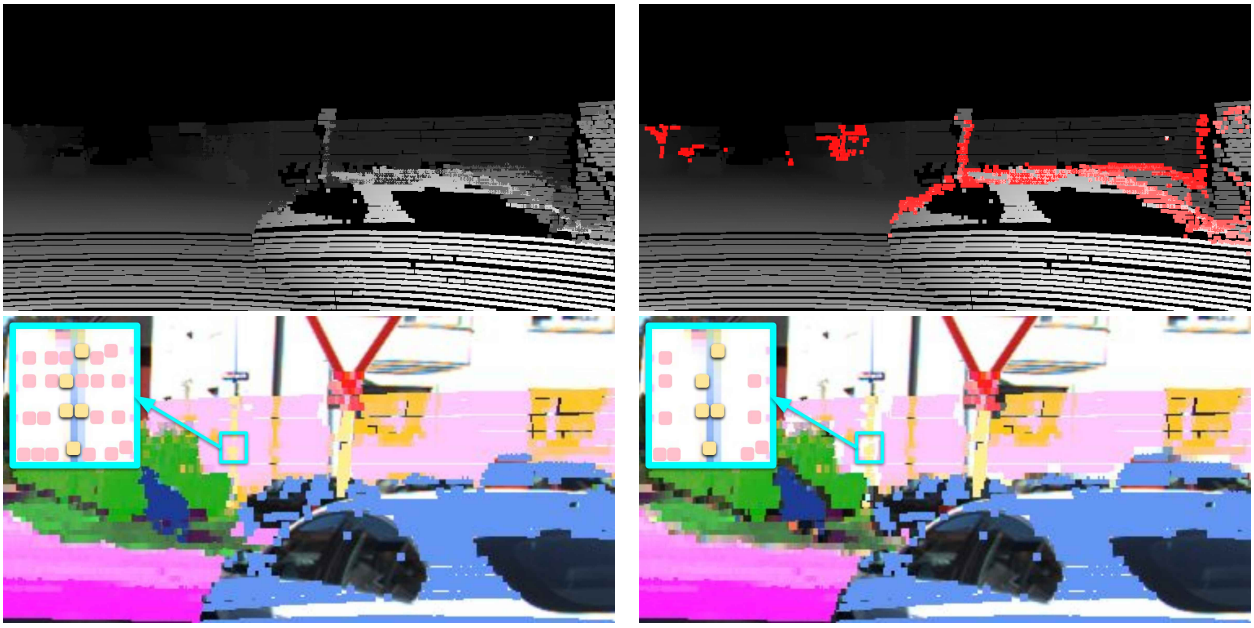


**Figure 7-6.** Top: Removing occluded but falsely visible points highlighted in red by disparity consistency check. Bottom: Removing the invading background points in pink for the thin object colored yellow by non-foreground suppression.

shows that some regions behind the car become visible in the converted depth map due to the alignment of different sensors. To address this issue, we follow Uhrig *et al*. [257] and use the same disparity methods for Cityscapes-DVPS to remove the sampled points that exhibit large relative errors, which are highlighted in red in the right figure. We refer to this processing as disparity consistency check. The second problem is that the regions of thin objects (*e.g*., poles) are usually invaded by the far-away background point cloud after projection. To alleviate this problem, for a small image patch, the projected background points are removed if there exists at least one foreground point that is closer to the camera. We refer to this processing as non-foreground suppression. In practice, we use a small $7 \times 7$ image patch. Doing so leaves clear boundaries for thin objects so they can be identified without confusion as shown in the second row of Fig. 7-6.

| DVPQ$^k_\lambda$ on Cityscapes-DVPS | k = 1 | | | k = 2 | | | k = 3 | | | k = 4 | | | Average | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ = 0.50 | 68.7 | 61.4 | 74.0 | 61.7 | 48.5 | 71.3 | 58.4 | 42.1 | 70.2 | 56.3 | 38.0 | 69.5 | 61.3 | 47.5 | 71.2 |
| $\lambda$ = 0.25 | 66.5 | 60.4 | 71.0 | 59.5 | 47.6 | 68.2 | 56.2 | 41.3 | 67.1 | 54.2 | 37.3 | 66.5 | 59.1 | 46.7 | 68.2 |
| $\lambda$ = 0.10 | 50.5 | 45.8 | 53.9 | 45.6 | 36.9 | 51.9 | 42.6 | 31.7 | 50.6 | 40.8 | 28.4 | 49.8 | 44.9 | 35.7 | 51.5 |
| Average | 61.9 | 55.9 | 66.3 | 55.6 | 44.3 | 63.8 | 52.4 | 38.4 | 62.6 | 50.4 | 34.6 | 61.9 | **55.1** | **43.3** | **63.6** |

| DVPQ$^k_\lambda$ on SemKITTI-DVPS | k = 1 | | | k = 5 | | | k = 10 | | | k = 20 | | | Average | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ = 0.50 | 54.7 | 46.4 | 60.6 | 51.5 | 41.0 | 59.1 | 50.1 | 38.5 | 58.5 | 49.2 | 36.9 | 58.2 | 51.4 | 40.7 | 59.1 |
| $\lambda$ = 0.25 | 52.0 | 44.8 | 57.3 | 48.8 | 39.4 | 55.7 | 47.4 | 37.0 | 55.1 | 46.6 | 35.6 | 54.7 | 48.7 | 39.2 | 55.7 |
| $\lambda$ = 0.10 | 40.0 | 34.7 | 43.8 | 37.1 | 30.3 | 42.0 | 35.8 | 28.3 | 41.2 | 34.5 | 26.5 | 40.4 | 36.8 | 30.0 | 41.9 |
| Average | 48.9 | 42.0 | 53.9 | 45.8 | 36.9 | 52.3 | 44.4 | 34.6 | 51.6 | 43.4 | 33.0 | 51.1 | **45.6** | **36.6** | **52.2** |

**Table 7-I.** ViP-DeepLab performance for the task of *Depth-aware Video Panoptic Segmentation* (DVPS) evaluated on Cityscapes-DVPS and SemKITTI-DVPS. Each cell shows DVPQ$^k_\lambda$ | DVPQ$^k_\lambda$-Thing | DVPQ$^k_\lambda$-Stuff where $\lambda$ is the threshold of relative depth error, and $k$ is the number of frames. Smaller $\lambda$ and larger $k$ correspond to a higher accuracy requirement.

## 7.5 Experiments

In this section, we first present our major results on the new task Depth-aware Video Panoptic Segmentation. Then, we show our method applied to three sub-tasks, including video panoptic segmentation [132], monocular depth estimation [87], and multi-object tracking and segmentation [261].

### 7.5.1 Depth-aware Video Panoptic Segmentation

Tab. 7-I shows our results on Depth-aware Video Panoptic Segmentation. We evaluate our method on the datasets Cityscapes-DVPS and SemKITTI-DVPS so that the research community can compare their methods with it. The evaluation is based on our proposed DVPQ$^k_\lambda$ metric (Eq. (7.5)), where $\lambda$ is the threshold of relative depth error, and $k$ denotes the length of the short video clip used in evaluation. The training time is about 1 day using 32 TPUs.

Following [132], we set $k = \{1, 2, 3, 4\}$ out of the total 6 frames per video sequence for Cityscapes-DVPS. By contrast, we set $k = \{1, 5, 10, 20\}$ for SemKITTI-DVPS which contains much longer video sequences, and we aim to evaluate a longer temporal consistency. We study the drops of DVPQ$^k_\lambda$ as the number of frames $k$ increases, where smaller performance drops indicate higher temporal consistency. Interestingly, as the number of frames $k$ increases, the

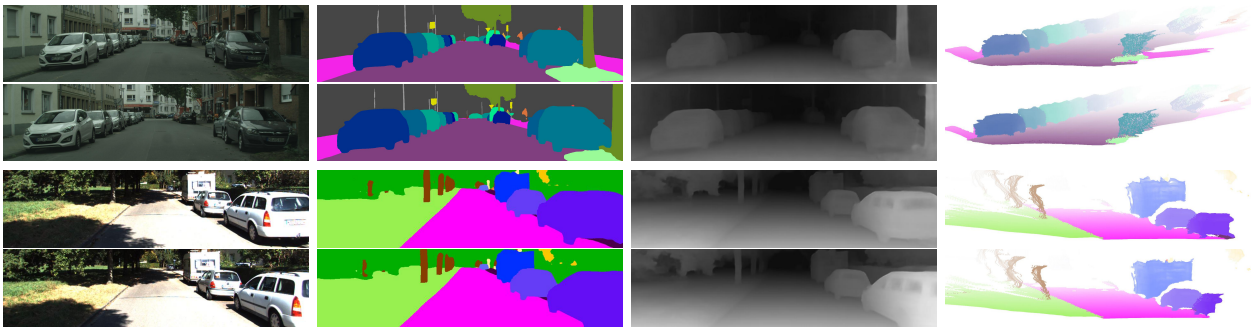


**Figure 7-7.** Prediction visualizations on Cityscapes-DVPS (top) and SemKITTI-DVPS (bottom). From left to right: input image, temporally consistent panoptic segmentation prediction, monocular depth prediction, and point cloud visualization.

performance drops on SemKITTI-DVPS are smaller than that on Cityscapes-DVPS. For example, DVPQ$^{1}_{0.5}$ - DVPQ$^{2}_{0.5}$ on Cityscapes-DVPS is $7\%$, while DVPQ$^{1}_{0.5}$ - DVPQ$^{5}_{0.5}$ on SemKITTI-DVPS is $3.2\%$. We speculate that this is because the annotation frame rate is higher on SemKITTI-DVPS (*cf*.only every 5th frame is annotated on Cityscapes-DVPS), making our ViP-DeepLab's offsets prediction easier for the following frames, despite the evaluation clip length $k$ is larger. At last, we use the mean of DVPS$^{k}_{\lambda}$ with different $\lambda$ and $k$ as the final performance score. Fig. 7-7 visualizes the predictions of our method on the validation set of Cityscapes-DVPS (top) and SemKITTI-DVPS (bottom), where the second column shows $P_t$ and $R_t$ defined in prediction stitching. Although the training samples of SemKITTI-DVPS are sparse points, our method is able to predict smooth and sharp predictions, as the points are evenly distributed in the regions covered by the Velodyne data. After experimenting on DVPS, we compare ViP-DeepLab with the previous state-of-the-arts on the sub-tasks to showcase its strong performance.

### 7.5.2 Video Panoptic Segmentation

The first sub-task of DVPS is Video Panoptic Segmentation (VPS). We conduct experiments on Cityscapes-VPS following the setting of [132] with the depth head removed. Tab. 7-II shows our major results on their validation set (top) and the test set where the annotations are missing (bottom). As the table shows, our method outperforms VPSNet [132] by 5.6% VPQ on the validation set and 5.1% VPQ on the test set.

| Val set | VPSNet [132] | | | ViP-DeepLab | | |
|---|---|---|---|---|---|---|
| k = 1 | 65.0 | 59.0 | 69.4 | 70.4 | 63.2 | 75.7 |
| k = 2 | 57.6 | 45.1 | 66.7 | 63.6 | 50.7 | 73.0 |
| k = 3 | 54.4 | 39.2 | 65.6 | 60.1 | 44.0 | 71.9 |
| k = 4 | 52.8 | 35.8 | 65.3 | 58.1 | 40.2 | 71.2 |
| VPQ | 57.5 | 44.8 | 66.7 | **63.1** | **49.5** | **73.0** |

| Test set | VPSNet [132] | | | ViP-DeepLab | | |
|---|---|---|---|---|---|---|
| k = 1 | 64.2 | 59.0 | 67.7 | 68.9 | 61.6 | 73.5 |
| k = 2 | 57.9 | 46.5 | 65.1 | 62.9 | 51.0 | 70.5 |
| k = 3 | 54.8 | 41.1 | 63.4 | 59.9 | 46.0 | 68.8 |
| k = 4 | 52.6 | 36.5 | 62.9 | 58.2 | 42.1 | 68.4 |
| VPQ | 57.4 | 45.8 | 64.8 | **62.5** | **50.2** | **70.3** |

**Table 7-II.** VPQ on Cityscapes-VPS. Each cell shows VPQ$^k$ | VPQ$^k$-Thing | VPQ$^k$-Stuff. VPQ is averaged over $k = \{1, 2, 3, 4\}$. $k = \{0, 5, 10, 15\}$ in [132] correspond to $k = \{1, 2, 3, 4\}$ in this chapter as we use different notations.

| Method | k = 1 | k = 2 | k = 3 | k = 4 | VPQ |
|---|---|---|---|---|---|
| Baseline | 65.7 | 58.9 | 55.8 | 53.6 | 58.5 |
| + MV | 66.7 | 59.3 | 56.1 | 54.1 | 59.0 |
| + CS | 67.9 | 60.4 | 56.8 | 54.7 | 59.9 |
| + DenseContext | 68.2 | 61.3 | 58.2 | 56.1 | 60.9 |
| + AutoAug [55] | 68.6 | 61.6 | 58.6 | 56.3 | 61.3 |
| + RFP [214] | 69.2 | 62.3 | 59.2 | 57.0 | 61.9 |
| + TTA | 70.3 | 63.2 | 59.9 | 57.5 | 62.7 |
| + SSL | 70.4 | 63.6 | 60.1 | 58.1 | 63.1 |

**Table 7-III.** Ablation Study on Cityscapes-VPS.

Tab. 7-III shows the ablation study on Cityscapes-VPS. The baseline is our method with backbone WR-41 [40], [277], [305] pre-trained on ImageNet [230]. Next, 'MV' initializes the model with a checkpoint pretrained on Mapillary Vistas [198]. 'CS' uses a model further pretrained on Cityscapes videos with pseudo labels [40] on the *train* sequence. Both 'MV' and 'CS' only involve image panoptic segmentation pretraining. Hence, they mainly improve image PQ (*i.e.*$k = 1$) but increases the gaps between VPQ$^k$ (*e.g.*, VPQ$^1$ - VPQ$^2$), showing the temporal consistency benefits less from the pretrained models. Then, 'DenseContext' increases the number of the context modules (from 1 to 4) for the next-frame instance branch, which narrows down the gaps between VPQ$^k$. 'AutoAug' uses AutoAugment [55] to augment the data.

| Method | Rank | SILog | sqRel | absRel | iRMSE |
|---|---|---|---|---|---|
| DORN [81] | 10 | 11.77 | 2.23 | 8.78 | 12.98 |
| BTS [157] | 9 | 11.67 | 2.21 | 9.04 | 12.23 |
| BANet [1] | 8 | 11.61 | 2.29 | 9.38 | 12.23 |
| MPSD [5] | 2 | 11.12 | 2.07 | 8.99 | 11.56 |
| ViP-DeepLab | 1 | 10.80 | 2.19 | 8.94 | 11.77 |

**Table 7-IV.** KITTI Depth Prediction Leaderboard. Ranking includes published and unpublished methods.

'RFP' adds Recursive Feature Pyramid (RFP) [214] to enhance the backbone. 'TTA' stands for test-time augmentation, which includes multi-scale inference at scales 0.5:1.75:0.25 and horizontal flipping. In 'SSL', we follow Naive-Student [40] to generate temporally consistent pseudo labels on the unlabeled *train* sequence in Cityscapes videos [53], which adds more training samples for temporal consistency, as demonstrated by +0.1% on $\text{VPQ}^1$ and +0.6% on $\text{VPQ}^4$.

### 7.5.3 Monocular Depth Estimation

The second sub-task of DVPS is monocular depth estimation. We test our method on the KITTI depth benchmark [257]. Tab. 7-IV shows the results on the leaderboard. Our model is pretrained on Mapillary Vistas [198] and Cityscapes videos with pseudo labels [40] (*i.e.*, the same pretrained checkpoint we used in the previous experiments). Then the model is fine-tuned with the training and validation set provided by KITTI depth benchmark [257]. However, the model is slightly different from the previous ones in the following aspects. It does not use RFP [214]. In TTA, it only has horizontal flipping. We use $\pm 5$ degrees of random rotation during training, which improves SILog by 0.27. The previous models use a decoder with stride 8 and 4. Here, we find it useful to further exploit decoder stride 2, which improves SILog by 0.17. After the above changes, our method achieves the best results on KITTI depth benchmark.

| Method | Pedestrians | | | Cars | | |
|---|---|---|---|---|---|---|
| | Rank | sMOTSA | MOTSA | Rank | sMOTSA | MOTSA |
| TrackR-CNN [261] | 20 | 47.3 | 66.1 | 19 | 67.0 | 79.6 |
| MOTSFusion [185] | 13 | 58.7 | 72.9 | 12 | 75.0 | 84.1 |
| PointTrack [292] | 11 | 61.5 | 76.5 | 5 | 78.5 | 90.9 |
| ReMOTS [295] | 6 | 66.0 | 81.3 | 9 | 75.9 | 86.7 |
| ViP-DeepLab | | 67.7 | 83.4 | | 80.6 | 90.3 |
| ViP-DeepLab + KF | 1 | 68.7 | 84.5 | 3 | 81.0 | 90.7 |

**Table 7-V.** KITTI MOTS Leaderboard. Ranking includes published and unpublished methods.

### 7.5.4 Multi-Object Tracking and Segmentation

Finally, we evaluate our method on the KITTI MOTS benchmark [261]. Tab. 7-V shows the leaderboard results. Different from the previous experiments, this benchmark only tracks pedestrians and cars. Adopting the same strategy as we used for Cityscapes-VPS, ViP-DeepLab outperforms all the published methods and achieves 67.7% and 80.6% sMOTSA for pedestrians and cars, respectively. To further improve our results, we use Kalman filter (KF) [271] to re-localize missing objects that are occluded or detection failures. This mechanism improves the sMOTSA by 1.0% and 0.4% for pedestrians and cars, respectively.

## 7.6 Conclusion

In this chapter, we propose a new challenging task Depth-aware Video Panoptic Segmentation, which combines monocular depth estimation and video panoptic segmentation, as a step towards solving the inverse projection problem in vision. For this task, we propose Depth-aware Video Panoptic Quality as the evaluation metric along with two derived datasets. We present ViP-DeepLab as a strong baseline for this task. Additionally, our ViP-DeepLab also achieves state-of-the-art performances on several sub-tasks, including monocular depth estimation, video panoptic segmentation, and multi-object tracking and segmentation.

# Chapter 8

# Conclusion

In this dissertation, I have presented a total of six research projects. They address the issues of data efficiency and model complexity of deep neural networks applied to solving various computer vision tasks, including image classification, object detection, instance segmentation, monocular depth estimation, video panoptic segmentation, *etc*. They also investigate different training scenarios, *e.g.* few-shot learning, semi-supervised learning, multi-task learning, *etc*.

Chapter 2 and 3 present our works on the data efficiency of deep neural networks. In Chapter 2, we study a novel setting of the few-shot learning problem where the model is evaluated for large-scale and few-shot learning at the same time. Motivated by the observation that the network parameters and the feature activations have highly similar structures in space, we propose a method to learn a category-agnostic mapping from activations to parameters. This mapping is used to predict parameters from activations instead of learning parameters from a limited number of training samples. It works well in the joint evaluation of large-scale and few-shot categories. In Chapter 3, we propose Deep Co-Training for semi-supervised image recognition. It extends the Co-Training framework to deep learning by training multiple neural networks as the views and adding divergence loss to encourage cross-view consistency. We also use adversarial training to force the views to be different and complementary to prevent the models from collapsing. This additional force that pushes models away is proven to be helpful and necessary, and improves the performance significantly.

Chapter 4, 5, 6, and 7 present our works on the model complexity of deep neural networks. In Chapter 4, we propose Gradually Updated Neural Network, an alternative method to cascading layers for increasing the depths of neural networks. It adds computation orderings to the channels of convolutional neural networks, and the outputs are computed gradually in order to increase the depth. The added orderings increase the representation capacity of convolutional neural networks without introducing additional computations while eliminating the harmful overlap singularities inherent in standard neural networks. In Chapter 5, we propose Neural Rejuvenation to improve the computational resource utilization of deep neural networks. It rejuvenates underutilized neurons during training by reallocating and reinitializing them. It is composed of three components: resource utilization monitoring, underutilized neuron rejuvenation, and training schemes for networks with mixed types of neurons. It enhances resource utilization, hence improving the performance of neural networks. In Chapter 6, we present DetectoRS for object detection, which is motivated by reusing computations to reduce the architecture complexity. It introduces two techniques: Recursive Feature Pyramid and Switchable Atrous Convolution. Recursive Feature Pyramid reuses the computation of the backbone and the feature pyramid to increase the representation capacity. Switchable Atrous Convolution reuses the computation of atrous convolution to grant the models ability to adapt to different object scales as needed. DetectoRS achieved state-of-the-arts on the benchmarks of object detection, instance segmentation, and panoptic segmentation at the same time. In Chapter 7, we study visual feature reusing for multi-task learning to reduce the model complexity. We propose ViP-DeepLab for the joint task of monocular depth estimation and video panoptic segmentation. It sets a strong baseline for the joint task while achieving state-of-the-art performance on several individual tasks, including monocular depth estimation, video panoptic segmentation, and multi-object tracking and segmentation.

The techniques introduced in this dissertation have impacts on both academia and real-world industrial applications. Further improvements could lie in many directions. For example, the experiments in this dissertation were mostly done on graphics processing units or tensor pro-

cessing units. Device-specific computations can utilize the computing devices more efficiently and achieve better performance. Stronger normalization methods or optimization techniques can also improve data efficiency. Data augmentation methods could be task-specific as well, encouraging the models to be more robust to changes such as rotation, object scales, lighting conditions, *etc.* Finally, self-supervised learning that requires no human annotations also has a strong impact on both academia and real-world applications. There are many interesting future directions and here we only list a few.

# References


[1] S. Aich, J. M. U. Vianney, M. A. Islam, M. Kaur, and B. Liu, "Bidirectional attention network for monocular depth estimation," *arXiv preprint arXiv:2009.00743*, 2020.

[2] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2270–2278.

[3] M. Amirul Islam, M. Rochan, N. D. Bruce, and Y. Wang, "Gated feedback refinement network for dense image labeling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3751–3759.

[4] A. Anandkumar and R. Ge, "Efficient approaches for escaping higher order saddle points in non-convex optimization," in *Proceedings of the Conference on Learning Theory*, 2016.

[5] M. L. Antequera, P. Gargallo, M. Hofinger, and S. Rota, "Mapillary planet-scale depth dataset," in *Proceedings of the European conference on computer vision (ECCV)*, 2020.

[6] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *CoRR*, vol. abs/1701.07875, 2017. arXiv: 1701.07875.

[7] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," *International Conference on Learning Representations*, 2018.

[8] A. Athar, S. Mahadevan, A. Ošep, L. Leal-Taixé, and B. Leibe, "Stem-seg: Spatio-temporal embeddings for instance segmentation in videos," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[9] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," in *Lazy learning*, Springer, 1997, pp. 75–113.

[10] J. Ba and R. Caruana, "Do deep nets really need to be deep?" In *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[11] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *arXiv preprint arXiv:1412.7755*, 2014.

[12] P. Bachman, O. Alsharif, and D. Precup, "Learning with pseudo-ensembles," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 3365–3373.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[14] M. Bai and R. Urtasun, "Deep watershed transform for instance segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[15] X. Bai, B. Wang, C. Yao, W. Liu, and Z. Tu, "Co-transduction for shape retrieval," *IEEE Transactions on Image Processing*, vol. 21, no. 5, pp. 2747–2757, May 2012. DOI: 10.1109/TIP.2011.2170082.

[16] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[17] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, 1981.

[18] D. M. Beck and S. Kastner, "Top-down and bottom-up mechanisms in biasing competition in the human brain," *Vision research*, vol. 49, no. 10, pp. 1154–1165, 2009.

[19] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "Semantickitti: A dataset for semantic scene understanding of lidar sequences," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[20] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," in *Proceedings of the IEEE international conference on computer vision*, 2019.

[21] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2016.

[22] P. Bloom, *How children learn the meanings of words*. MIT press, 2002.

[23] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[24] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-nms–improving object detection with one line of code," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5561–5569.

[25] U. Bonde, P. F. Alcantarilla, and S. Leutenegger, "Towards bounding-box free panoptic segmentation," *arXiv preprint arXiv:2002.07705*, 2020.

[26] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996. DOI: 10.1007/BF00058655.

[27] L. Breiman, "Statistical modeling: The two cultures (with comments and a rejoinder by the author)," *Statist. Sci.*, vol. 16, no. 3, pp. 199–231, Aug. 2001. DOI: 10.1214/ss/1009213726.

[28] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," *arXiv preprint arXiv:2102.06171*, 2021.

[29] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," AAAI, 2018.

[30] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *Proceedings of the European Conference on Computer Vision*, Springer, 2016, pp. 354–370.

[31] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6154–6162.

[32] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, *et al.*, "Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2956–2964.

[33] J. Cao, H. Cholakkal, R. M. Anwer, F. S. Khan, Y. Pang, and L. Shao, "D2det: Towards high quality object detection and instance segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 485–11 494.

[34] Y. Cao, Z. Wu, and C. Shen, "Estimating depth from monocular images as classification using deep fully convolutional residual networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 11, 2017.

[35] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *arXiv preprint arXiv:1807.05520*, 2018.

[36] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang, *et al.*, "Hybrid task cascade for instance segmentation," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019, pp. 4974–4983.

[37] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[38] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, and R. Nevatia, "Abc-cnn: An attention based convolutional neural network for visual question answering," *arXiv preprint arXiv:1511.05960*, 2015.

[39] L.-C. Chen, A. Hermans, G. Papandreou, F. Schroff, P. Wang, and H. Adam, "Masklab: Instance segmentation by refining object detection with semantic and direction features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4013–4022.

[40] L.-C. Chen, R. G. Lopes, B. Cheng, M. D. Collins, E. D. Cubuk, B. Zoph, H. Adam, and J. Shlens, "Naive-student: Leveraging semi-supervised learning in video sequences for urban scene segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2020.

[41] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[42] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *Proceedings of the International Conference on Learning Representations*, 2015.

[43] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[44] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[45] Q. Chen, A. Cheng, X. He, P. Wang, and J. Cheng, "Spatialflow: Bridging all tasks for panoptic segmentation," *arXiv preprint arXiv:1910.08787*, 2019.

[46] W. Chen, Z. Fu, D. Yang, and J. Deng, "Single-image depth perception in the wild," in *Advances in neural information processing systems*, 2016.

[47] Y. Chen, G. Lin, S. Li, O. Bourahla, Y. Wu, F. Wang, J. Feng, M. Xu, and X. Li, "Banet: Bidirectional aggregation network with occlusion handling for panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3793–3802.

[48] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic convolution: Attention over convolution kernels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[49] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," *CoRR*, vol. abs/1707.01629, 2017. arXiv: 1707.01629.

[50] Z. Chen and B. Liu, "Mining topics in documents: Standing on the shoulders of big data," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, 2014, pp. 1116–1125. DOI: 10.1145/2623330.2623622.

[51] Z. Chen and B. Liu, "Topic modeling using topics from many domains, lifelong learning and big data," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 703–711.

[52] B. Cheng, M. D. Collins, Y. Zhu, T. Liu, T. S. Huang, H. Adam, and L.-C. Chen, "Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[53] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[54] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[55] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[56] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 764–773.

[57] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. Salakhutdinov, "Good semi-supervised learning that requires a bad GAN," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 6513–6523.

[58] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Jun. 2005. DOI: 10.1109/CVPR.2005.177.

[59] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[60] R. Desimone, "Visual attention mediated by biased competition in extrastriate visual cortex," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 353, no. 1373, pp. 1245–1255, 1998.

[61] R. Desimone and J. Duncan, "Neural mechanisms of selective visual attention," *Annual review of neuroscience*, vol. 18, no. 1, pp. 193–222, 1995.

[62] R. Diaz and A. Marathe, "Soft labels for ordinal regression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4738–4747.

[63] H. Ding, S. Qiao, A. Yuille, and W. Shen, “Deeply shape-guided cascade for instance segmentation,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[64] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “Decaf: A deep convolutional activation feature for generic visual recognition,” in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 647–655.

[65] Y. Dong and E. J. Nestler, “The neural rejuvenation hypothesis of cocaine addiction,” *Trends Pharmacol Sci*, vol. 35, no. 8, pp. 374–383, Aug. 2014. DOI: 10.1016/j.tips.2014.05.005.

[66] X. Du, T.-Y. Lin, P. Jin, G. Ghiasi, M. Tan, Y. Cui, Q. V. Le, and X. Song, “Spinenet: Learning scale-permuted backbone for recognition and localization,” *arXiv preprint arXiv:1912.05027*, 2019.

[67] D. Eigen and R. Fergus, “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture,” in *Proceedings of the IEEE international conference on computer vision*, 2015.

[68] D. Eigen, C. Puhrsch, and R. Fergus, “Depth map prediction from a single image using a multi-scale deep network,” in *Advances in neural information processing systems*, 2014.

[69] T. Elsken, J.-H. Metzen, and F. Hutter, “Simple and efficient architecture search for convolutional neural networks,” *arXiv preprint arXiv:1711.04528*, 2017.

[70] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*.

[71] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge: A retrospective,” *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015.

[72] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[73] K. Fang, Y. Xiang, X. Li, and S. Savarese, “Recurrent autoregressive networks for online multi-object tracking,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2018.

[74] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[75] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Detect to track and track to detect,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[76] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, 2009.

[77] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1126–1135.

[78] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach*. Pearson, 2012.

[79] S. C. Fralick, “Learning to recognize patterns without a teacher,” *IEEE Trans. Information Theory*, vol. 13, no. 1, pp. 57–64, 1967. DOI: 10.1109/TIT.1967.1053952.

[80] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Training pruned neural networks,” *arXiv preprint arXiv:1803.03635*, 2018.

[81] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, “Deep ordinal regression network for monocular depth estimation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[82] Y. Gan, X. Xu, W. Sun, and L. Lin, “Monocular depth estimation with affinity, vertical pooling, and label enhancement,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[83] N. Gao, Y. Shan, Y. Wang, X. Zhao, Y. Yu, M. Yang, and K. Huang, “Ssap: Single-shot instance segmentation with affinity pyramid,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 642–651.

[84] R. Garg, V. K. Bg, G. Carneiro, and I. Reid, “Unsupervised cnn for single view depth estimation: Geometry to the rescue,” in *European conference on computer vision*, Springer, 2016.

[85] X. Gastaldi, “Shake-shake regularization,” *CoRR*, vol. abs/1705.07485, 2017. arXiv: 1705.07485.

[86] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *CoRR*, vol. abs/1508.06576, 2015. arXiv: 1508.06576.

[87] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012.

[88] G. Ghiasi, T.-Y. Lin, and Q. V. Le, “Nas-fpn: Learning scalable feature pyramid architecture for object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7036–7045.

[89] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[90] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014*, 2014. DOI: 10.1109/CVPR.2014.81.

[91] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[92] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, “Digging into self-supervised monocular depth estimation,” in *Proceedings of the IEEE international conference on computer vision*, 2019.

[93] S. D. Goggin, K. M. Johnson, and K. E. Gustafson, “A second-order translation, rotation and scale invariant neural network,” in *Advances in neural information processing systems*, 1991, pp. 313–319.

[94] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *International Conference on Learning Representations, ICLR, 2015*, 2015.

[95] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, “Morphnet: Fast & simple resource-constrained structure learning of deep networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[96] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, “Accurate, large minibatch sgd: Training imagenet in 1 hour,” *arXiv preprint arXiv:1706.02677*, 2017.

[97] G. Griffin, A. Holub, and P. Perona, “Caltech-256 object category dataset,” in *Technical Report 7694, California Institute of Technology*, 2007.

[98] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, “Cascade cost volume for high-resolution multi-view stereo and stereo matching,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2495–2504.

[99] C. Guo, B. Fan, Q. Zhang, S. Xiang, and C. Pan, “Augfpn: Improving multi-scale feature learning for object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 595–12 604.

[100] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, “Eie: Efficient inference engine on compressed deep neural network,” in *Computer Architecture (ISCA), 2016*, IEEE, 2016, pp. 243–254.

[101] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[102] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[103] Y. Hao, Y. Zhang, K. Liu, S. He, Z. Liu, H. Wu, and J. Zhao, “An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge,” in *Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2017, pp. 221–231.

[104] B. Hariharan, P. A. Arbeláez, R. B. Girshick, and J. Malik, “Hypercolumns for object segmentation and fine-grained localization,” in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 447–456. DOI: 10.1109/CVPR.2015.7298642.

[105] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, “Simultaneous detection and segmentation,” in *Proceedings of the European Conference on Computer Vision*, Springer, 2014, pp. 297–312.

[106] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in *Advances in neural information processing systems*, 1993, pp. 164–171.

[107] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[108] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[109] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *ECCV*, 2016.

[110] X. He, R. S. Zemel, and M. Á. Carreira-Perpiñán, “Multiscale conditional random fields for image labeling,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[111] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 1398–1406. DOI: 10.1109/ICCV.2017.155.

[112] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[113] H. Hirschmuller, “Stereo processing by semiglobal matching and mutual information,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2007.

[114] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[115] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, “A real-time algorithm for signal analysis with the help of the wavelet transform,” in *Wavelets: Time-Frequency Methods and Phase Space*, Springer Berlin Heidelberg, 1989, pp. 289–297.

[116] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[117] I. P. Howard, *Perceiving in depth, volume 1: Basic mechanisms*. Oxford University Press, 2012.

[118] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” *CoRR*, vol. abs/1709.01507, 2017. arXiv: 1709.01507.

[119] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, “Snapshot ensembles: Train 1, get M for free,” *CoRR*, vol. abs/1704.00109, 2017.

[120] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[121] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, “Deep networks with stochastic depth,” *CoRR*, vol. abs/1603.09382, 2016.

[122] M. Huh, P. Agrawal, and A. A. Efros, “What makes imagenet good for transfer learning?” *CoRR*, vol. abs/1608.08614, 2016.

[123] H. J. S. III, “Probability of error of some adaptive pattern-recognition machines,” *IEEE Trans. Information Theory*, vol. 11, no. 3, pp. 363–371, 1965. DOI: 10.1109/TIT.1965.1053799.

[124] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.

[125] C. Jiang, H. Xu, W. Zhang, X. Liang, and Z. Li, “Sp-nas: Serial-to-parallel backbone search for object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 863–11 872.

[126] L. Jin, J. Lazarow, and Z. Tu, “Introspective classification with convolutional nets,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 823–833.

[127] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with gpus,” *arXiv preprint arXiv:1702.08734*, 2017.

[128] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache, “Learning visual features from large weakly supervised data,” in *ECCV*, Springer, 2016, pp. 67–84.

[129] A. Kazemy, S. A. Hosseini, and M. Farrokhi, “Second order diagonal recurrent neural network,” in *Industrial Electronics, ISIE 2007.*, IEEE, 2007, pp. 251–256.

[130] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[131] M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres, "Efficient decomposition of image and mesh graphs by lifted multicuts," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[132] D. Kim, S. Woo, J.-Y. Lee, and I. S. Kweon, "Video panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9859–9868.

[133] J. Kim, J. Kwon Lee, and K. Mu Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1637–1645.

[134] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[135] A. Kirillov, R. Girshick, K. He, and P. Dollár, "Panoptic feature pyramid networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6399–6408.

[136] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, "Panoptic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9404–9413.

[137] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML Deep Learning workshop*, 2015.

[138] I. Kokkinos, "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6129–6138.

[139] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," in *Proceedings of the European Conference on Computer Vision*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., 2020, pp. 491–507.

[140] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò, "Deep neural decision forests," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1467–1475. DOI: 10.1109/ICCV.2015.172.

[141] A. Krizhevsky, "Learning multiple layers of features from tiny images.," in *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[142] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[143] Y. Kuznietsov, J. Stuckler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[144] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *International Conference on Learning Representations, ICLR, 2017*, 2017.

[145] B. M. Lake, R. Salakhutdinov, J. Gross, and J. B. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the 33th Annual Meeting of the Cognitive Science Society, CogSci 2011, Boston, Massachusetts, USA, July 20-23, 2011*, 2011.

[146] B. M. Lake, R. R. Salakhutdinov, and J. Tenenbaum, "One-shot learning by inverting a compositional causal process," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 2526–2534.

[147] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[148] G. Larsson, M. Maire, and G. Shakhnarovich, “Fractalnet: Ultra-deep neural networks without residuals,” *CoRR*, vol. abs/1605.07648, 2016.

[149] H. Law and J. Deng, “Cornernet: Detecting objects as paired keypoints,” in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 734–750.

[150] J. Lazarow, K. Lee, K. Shi, and Z. Tu, “Learning instance occlusion for panoptic segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 720–10 729.

[151] L. Leal-Taixé, C. Canton-Ferrer, and K. Schindler, “Learning by tracking: Siamese cnn for robust target association,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016.

[152] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, “Speeding-up convolutional neural networks using fine-tuned cp-decomposition,” *arXiv preprint arXiv:1412.6553*, 2014.

[153] V. Lebedev and V. Lempitsky, “Fast convnets using group-wise brain damage,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016, pp. 2554–2564.

[154] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[155] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Advances in neural information processing systems*, 1990, pp. 598–605.

[156] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, “Deeply-supervised nets,” in *Artificial Intelligence and Statistics*, 2015, pp. 562–570.

[157] J. H. Lee, M.-K. Han, D. W. Ko, and I. H. Suh, “From big to small: Multi-scale local planar guidance for monocular depth estimation,” *arXiv preprint arXiv:1907.10326*, 2019.

[158] K.-H. Lee, X. Chen, G. Hua, H. Hu, and X. He, “Stacked cross attention for image-text matching,” *arXiv preprint arXiv:1803.08024*, 2018.

[159] B. Leibe, A. Leonardis, and B. Schiele, “Combined object categorization and segmentation with an implicit shape model,” in *Workshop on statistical learning in computer vision, ECCV*, 2004.

[160] B. Li, C. Shen, Y. Dai, A. Van Den Hengel, and M. He, “Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1119–1127.

[161] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv preprint arXiv:1608.08710*, 2016.

[162] H. Li, Z. Wu, C. Zhu, C. Xiong, R. Socher, and L. S. Davis, “Learning from noisy anchors for one-stage object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 588–10 597.

[163] J. Li, A. Raventos, A. Bhargava, T. Tagawa, and A. Gaidon, “Learning to fuse things and stuff,” *arXiv preprint arXiv:1812.01192*, 2018.

[164] Q. Li, X. Qi, and P. H. Torr, “Unifying training and inference for panoptic segmentation,” *arXiv preprint arXiv:2001.04982*, 2020.

[165] X. Li, W. Wang, X. Hu, and J. Yang, “Selective kernel networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 510–519.

[166] Y. Li, Y. Chen, N. Wang, and Z. Zhang, "Scale-aware trident networks for object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6054–6063.

[167] Y. Li, X. Chen, Z. Zhu, L. Xie, G. Huang, D. Du, and X. Wang, "Attention-guided unified network for panoptic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[168] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3367–3375.

[169] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2181–2191.

[170] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013.

[171] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[172] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2980–2988.

[173] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[174] X. Lin, H. Wang, Z. Li, Y. Zhang, A. Yuille, and T. S. Lee, "Transfer of view-manifold learning to similarity perception of novel objects," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[175] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

[176] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[177] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8759–8768.

[178] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision*, Springer, 2016, pp. 21–37.

[179] Y. Liu, S. Yang, B. Li, W. Zhou, J. Xu, H. Li, and Y. Lu, "Affinity derivation and graph merge for instance segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[180] Y. Liu, Y. Wang, S. Wang, T. Liang, Q. Zhao, Z. Tang, and H. Ling, "Cbnet: A novel composite backbone network architecture for object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[181] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[182] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015. DOI: 10.1109/CVPR.2015.7298965.

[183] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” *arXiv preprint arXiv:1608.03983*, 2016.

[184] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. DOI: 10.1023/B:VISI.0000029664.99615.94.

[185] J. Luiten, T. Fischer, and B. Leibe, “Track to reconstruct and reconstruct to track,” *IEEE Robotics and Automation Letters*, 2020.

[186] J.-H. Luo, J. Wu, and W. Lin, “Thinet: A filter level pruning method for deep neural network compression,” *arXiv preprint arXiv:1707.06342*, 2017.

[187] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. 11, pp. 2579–2605, 2008.

[188] R. Mahjourian, M. Wicke, and A. Angelova, “Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[189] J. Mao, X. Wei, Y. Yang, J. Wang, Z. Huang, and A. L. Yuille, “Learning like a child: Fast novel visual concept learning from sentence descriptions of images,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2533–2541.

[190] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille, “Deep captioning with multimodal recurrent neural networks (m-rnn),” *CoRR*, vol. abs/1412.6632, 2014. arXiv: 1412.6632.

[191] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, *et al.*, “Evolving deep neural networks,” in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, Elsevier, 2019, pp. 293–312.

[192] T. Miyato, S. Maeda, M. Koyama, and S. Ishii, “Virtual adversarial training: A regularization method for supervised and semi-supervised learning,” *CoRR*, vol. abs/1704.03976, 2017. arXiv: 1704.03976.

[193] V. Mnih, N. Heess, A. Graves, *et al.*, “Recurrent models of visual attention,” in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[194] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *arXiv preprint arXiv:1611.06440*, 2016.

[195] A. Mordvintsev, C. Olah, and M. Tyka, “Deepdream - a code example for visualizing neural networks,” *Google Research*, 2015.

[196] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 2010.

[197] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[198] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder, “The mapillary vistas dataset for semantic understanding of street scenes,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[199] D. Neven, B. D. Brabandere, M. Proesmans, and L. V. Gool, "Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[200] K. Nigam and R. Ghani, "Analyzing the effectiveness and applicability of co-training," in *Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 6-11, 2000*, 2000, pp. 86–93. DOI: 10.1145/354756.354805.

[201] A. E. Orhan and X. Pitkow, "Skip connections eliminate singularities," *International Conference on Learning Representations*, 2018.

[202] S. E. Palmer, *Vision science: Photons to phenomenology*. MIT press, 1999.

[203] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, 2005.

[204] J. Pang, K. Chen, J. Shi, H. Feng, W. Ouyang, and D. Lin, "Libra r-cnn: Towards balanced learning for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 821–830.

[205] G. Papandreou, I. Kokkinos, and P.-A. Savalle, "Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[206] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun, "Megdet: A large mini-batch object detector," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 6181–6189.

[207] J. Peng, C. Wang, F. Wan, Y. Wu, Y. Wang, Y. Tai, C. Wang, J. Li, F. Huang, and Y. Fu, "Chained-tracker: Chaining paired attentive regression results for end-to-end joint multiple-object detection and tracking," in *European Conference on Computer Vision*, Springer, 2020, pp. 145–161.

[208] M. Pezeshki, L. Fan, P. Brakel, A. C. Courville, and Y. Bengio, "Deconstructing the ladder network architecture," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016, pp. 2368–2376.

[209] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[210] Z. Pizlo, "Perception viewed as an inverse problem," *Vision research*, vol. 41, no. 24, 2001.

[211] L. Porzi, S. R. Bulo, A. Colovic, and P. Kontschieder, "Seamless scene segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8277–8286.

[212] L. Porzi, M. Hofinger, I. Ruiz, J. Serrat, S. R. Bulo, and P. Kontschieder, "Learning multi-object tracking and segmentation from automatic annotations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6846–6855.

[213] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin, "Variational autoencoder for deep learning of images, labels and captions," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 2352–2360.

[214] S. Qiao, L.-C. Chen, and A. Yuille, "Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[215] S. Qiao, Z. Lin, J. Zhang, and A. L. Yuille, “Neural rejuvenation: Improving deep network training by enhancing computational resource utilization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 61–71.

[216] S. Qiao, C. Liu, W. Shen, and A. L. Yuille, “Few-shot image recognition by predicting parameters from activations,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7229–7238.

[217] S. Qiao, W. Shen, W. Qiu, C. Liu, and A. L. Yuille, “Scalenet: Guiding object proposal generation in supermarkets and beyond,” in *2017 IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29*, 2017.

[218] S. Qiao, W. Shen, Z. Zhang, B. Wang, and A. Yuille, “Deep co-training for semi-supervised image recognition,” in *Proceedings of the european conference on computer vision (eccv)*, 2018, pp. 135–152.

[219] S. Qiao, Z. Zhang, W. Shen, B. Wang, and A. Yuille, “Gradually updated neural networks for large-scale image recognition,” in *International Conference on Machine Learning*, PMLR, 2018, pp. 4188–4197.

[220] S. Qiao, Y. Zhu, H. Adam, A. Yuille, and L.-C. Chen, “Vip-deeplab: Learning visual perception with depth-aware video panoptic segmentation,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[221] W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. S. Kim, and Y. Wang, “Unrealcv: Virtual worlds for computer vision,” in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1221–1224.

[222] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, “Semi-supervised learning with ladder networks,” in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 3546–3554.

[223] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.

[224] S. Ravi and H. Larochelle, “Optimization as a model for few-shot learning,” in *5th International Conference on Learning Representations (ICLR)*, 2017.

[225] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, “Large-scale evolution of image classifiers,” *arXiv preprint arXiv:1703.01041*, 2017.

[226] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7263–7271.

[227] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

[228] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.

[229] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” *CoRR*, vol. abs/1412.6550, 2014. arXiv: 1412.6550.

[230] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[231] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 1163–1171.

[232] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 2226–2234.

[233] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, p. 901.

[234] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," in *Advances in neural information processing systems*, 2006.

[235] S. Schulter, P. Vernaza, W. Choi, and M. Chandraker, "Deep network flow for multi-object tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[236] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[237] S. Sharma, J. A. Ansari, J. K. Murthy, and K. M. Krishna, "Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018.

[238] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang, "Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015. DOI: 10.1109/CVPR.2015.7299024.

[239] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[240] B. Singh and L. S. Davis, "An analysis of scale invariance in object detection snip," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3578–3587.

[241] B. Singh, M. Najibi, and L. S. Davis, "Sniper: Efficient multi-scale training," in *Advances in Neural Information Processing Systems*, 2018, pp. 9310–9320.

[242] K. Sofiiuk, O. Barinova, and A. Konushin, "Adaptis: Adaptive instance selection network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7355–7363.

[243] G. Song, Y. Liu, and X. Wang, "Revisiting the sibling head in object detector," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[244] J. C. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005, vol. 65.

[245] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014.

[246] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Training very deep networks,” in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

[247] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[248] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[249] Y. Tai, J. Yang, and X. Liu, “Image super-resolution via deep recursive residual network,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3147–3155.

[250] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 6105–6114.

[251] M. Tan, R. Pang, and Q. V. Le, “Efficientdet: Scalable and efficient object detection,” *arXiv preprint arXiv:1911.09070*, 2019.

[252] S. Tang, M. Andriluka, B. Andres, and B. Schiele, “Multiple people tracking by lifted multicut and person re-identification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[253] A. Tarvainen and H. Valpola, “Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 1195–1204.

[254] Z. Tian, C. Shen, H. Chen, and T. He, “Fcos: Fully convolutional one-stage object detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9627–9636.

[255] Z. Tu, “Learning generative models via discriminative approaches,” in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383035.

[256] J. Uhrig, E. Rehder, B. Fröhlich, U. Franke, and T. Brox, “Box2pix: Single-shot instance segmentation by assigning pixels to object boxes,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018.

[257] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, “Sparsity invariant cnns,” in *2017 international conference on 3D Vision (3DV)*, IEEE, 2017.

[258] S. Vandenhende, S. Georgoulis, and L. Van Gool, “Mti-net: Multi-scale task interaction networks for multi-task learning,” in *European Conference on Computer Vision*, 2020.

[259] L. Vincent and P. Soille, “Watersheds in digital spaces: An efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 1991.

[260] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, “Matching networks for one shot learning,” in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 3630–3638.

[261] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "Mots: Multi-object tracking and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019.

[262] C. Wang, J. Miguel Buenaposada, R. Zhu, and S. Lucey, "Learning depth from monocular videos using direct methods," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[263] H. Wang, R. Luo, M. Maire, and G. Shakhnarovich, "Pixel consensus voting for panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[264] H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, and L.-C. Chen, "Axial-deeplab: Stand-alone axial-attention for panoptic segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[265] N. Wang, Y. Gao, H. Chen, P. Wang, Z. Tian, C. Shen, and Y. Zhang, "Nas-fcos: Fast neural architecture search for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 943–11 951.

[266] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille, "Towards unified depth and semantic prediction from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[267] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 2794–2802. DOI: 10.1109/ICCV.2015.320.

[268] X. Wang, S. Zhang, Z. Yu, L. Feng, and W. Zhang, "Scale-equalizing pyramid convolution for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[269] Y. Wang, L. Xie, C. Liu, S. Qiao, Y. Zhang, W. Zhang, Q. Tian, and A. Yuille, "SORT: Second-Order Response Transform for Visual Recognition," *International Conference on Computer Vision*, 2017.

[270] Y. Wang, L. Xie, S. Qiao, Y. Zhang, W. Zhang, and A. L. Yuille, "Multi-scale spatially-asymmetric recalibration for image classification," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 509–525.

[271] Z. Wang, L. Zheng, Y. Liu, and S. Wang, "Towards real-time multi-object tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2020.

[272] H. Wei, J. Zhang, F. Cousseau, T. Ozeki, and S. Amari, "Dynamics of learning near singularities in layered networks," *Neural Computation*, vol. 20, no. 3, pp. 813–843, 2008.

[273] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[274] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*, IEEE, 2017.

[275] Y. Wu, G. Zhang, Y. Gao, X. Deng, K. Gong, X. Liang, and L. Lin, "Bidirectional graph reasoning network for panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[276] Y. Wu, Y. Chen, L. Yuan, Z. Liu, L. Wang, H. Li, and Y. Fu, "Rethinking classification and localization for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 186–10 195.

[277] Z. Wu, C. Shen, and A. Van Den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," *Pattern Recognition*, 2019.

[278] R. Xia, C. Wang, X. Dai, and T. Li, "Co-training for semi-supervised sentiment classification based on dual-view bags-of-words representation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, 2015, pp. 1054–1063.

[279] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. L. Yuille, "Adversarial examples for semantic segmentation and object detection," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 1378–1387. DOI: 10.1109/ICCV.2017.153.

[280] J. Xie, R. Girshick, and A. Farhadi, "Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks," in *European Conference on Computer Vision*, Springer, 2016.

[281] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

[282] S. Xie and Z. Tu, "Holistically-nested edge detection," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015. DOI: 10.1109/ICCV.2015.164.

[283] Y. Xiong, R. Liao, H. Zhao, R. Hu, M. Bai, E. Yumer, and R. Urtasun, "Upsnet: A unified panoptic segmentation network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8818–8826.

[284] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *CoRR*, vol. abs/1304.5634, 2013. arXiv: 1304.5634.

[285] D. Xu, W. Ouyang, X. Wang, and N. Sebe, "Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[286] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe, "Multi-scale continuous crfs as sequential deep networks for monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[287] H. Xu, L. Yao, W. Zhang, X. Liang, and Z. Li, "Auto-fpn: Automatic network architecture adaptation for object detection beyond classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6649–6658.

[288] H. Xu and K. Saenko, "Ask, attend and answer: Exploring question-guided spatial attention for visual question answering," in *European Conference on Computer Vision*, Springer, 2016, pp. 451–466.

[289] J. Xu, Y. Cao, Z. Zhang, and H. Hu, "Spatial-temporal relation networks for multi-object tracking," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[290] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[291] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans, "Maximum margin clustering," in *Advances in neural information processing systems*, 2005, pp. 1537–1544.

[292] Z. Xu, W. Zhang, X. Tan, W. Yang, H. Huang, S. Wen, E. Ding, and L. Huang, "Segment as points for efficient online multi-object tracking and segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2020.

[293] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, "Condconv: Conditionally parameterized convolutions for efficient inference," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019, pp. 1307–1318.

[294] C. Yang, L. Xie, S. Qiao, and A. Yuille, "Knowledge distillation in generations: More tolerant teachers educate better students," *AAAI*, 2018.

[295] F. Yang, X. Chang, C. Dang, Z. Zheng, S. Sakti, S. Nakamura, and Y. Wu, "Remots: Self-supervised refining multi-object tracking and segmentation," *arXiv preprint arXiv:2007.03200*, 2020.

[296] M. Yang, K. Yu, C. Zhang, Z. Li, and K. Yang, "Denseaspp for semantic segmentation in street scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3684–3692.

[297] S. Yang and D. Ramanan, "Multi-scale recognition with dag-cnns," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1215–1223. DOI: 10.1109/ICCV.2015.144.

[298] T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L.-C. Chen, "Deeperlab: Single-shot image parser," *arXiv preprint arXiv:1902.05093*, 2019.

[299] Y. Yang, H. Li, X. Li, Q. Zhao, J. Wu, and Z. Lin, "Sognet: Scene overlap graph network for panoptic segmentation," *arXiv preprint arXiv:1911.07527*, 2019.

[300] J. Ye, X. Lu, Z. L. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," *CoRR*, vol. abs/1802.00124, 2018. arXiv: 1802.00124.

[301] W. Yin, Y. Liu, C. Shen, and Y. Yan, "Enforcing geometric constraints of virtual normal for depth prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[302] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[303] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *arXiv preprint arXiv:1812.08928*, 2018.

[304] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," *Preprint at https://arxiv.org/abs/1711.05908*, 2017.

[305] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference*, 2016.

[306] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[307] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, "Ga-net: Guided aggregation net for end-to-end stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 185–194.

[308] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," *CoRR*, vol. abs/1710.09412, 2017. arXiv: 1710.09412.

[309] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 4203–4212.

[310] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "Fairmot: On the fairness of detection and re-identification in multiple object tracking," *arXiv preprint arXiv:2004.01888*, 2020.

[311] Y. Zhang, Y. Chen, X. Bai, S. Yu, K. Yu, Z. Li, and K. Yang, "Adaptive unimodal cost volume filtering for deep stereo matching.," in *AAAI*, 2020, pp. 12 926–12 934.

[312] Z. Zhang, D. Cheng, X. Zhu, S. Lin, and J. Dai, "Integrated object detection and tracking with tracklet-conditioned detection," *arXiv preprint arXiv:1811.11167*, 2018.

[313] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille, "Single-shot object detection with enriched semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5813–5821.

[314] Z. Zhang, W. Shen, S. Qiao, Y. Wang, B. Wang, and A. Yuille, "Robust face detection via learning small faces on hard images," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 1361–1370.

[315] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2det: A single-shot object detector based on multi-level feature pyramid network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9259–9266.

[316] Z. Zhong, J. Yan, and C.-L. Liu, "Practical network blocks design with q-learning," *arXiv preprint arXiv:1708.05552*, 2017.

[317] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *European Conference on Computer Vision*, Springer, 2016, pp. 662–677.

[318] P. Zhou, B. Ni, C. Geng, J. Hu, and Y. Xu, "Scale-transferrable object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 528–537.

[319] X. Zhou, V. Koltun, and P. Krähenbühl, "Tracking objects as points," *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[320] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[321] X. Zhou, J. Zhuo, and P. Krahenbuhl, "Bottom-up object detection by grouping extreme and center points," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 850–859.

[322] Y. Zhou, L. Xie, W. Shen, Y. Wang, E. K. Fishman, and A. L. Yuille, "A fixed-point model for pancreas segmentation in abdominal ct scans," in *International conference on medical image computing and computer-assisted intervention*, Springer, 2017, pp. 693–701.

[323] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Press, 2012.

[324] Z.-H. Zhou and M. Li, "Tri-training: Exploiting unlabeled data using three classifiers," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 11, pp. 1529–1541, Nov. 2005. DOI: 10.1109/TKDE.2005.186.

[325] Z. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: Many could be better than all," *Artif. Intell.*, vol. 137, no. 1-2, pp. 239–263, 2002. DOI: 10.1016/S0004-3702(02)00190-X.

[326] C. Zhu, Y. He, and M. Savvides, “Feature selective anchor-free module for single-shot object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 840–849.

[327] J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M.-H. Yang, “Online multi-object tracking with dual matching attention networks,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[328] X. J. Zhu, “Semi-supervised learning literature survey,” 2005.

[329] X. Zhu, H. Hu, S. Lin, and J. Dai, “Deformable convnets v2: More deformable, better results,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9308–9316.

[330] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.

[331] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” *arXiv preprint arXiv:1707.07012*, vol. 2, no. 6, 2017.

# Vita

Siyuan Qiao is completing his Doctor of Philosophy degree in the Department of Computer Science, Johns Hopkins University, under the supervision of Bloomberg Distinguished Professor Alan L. Yuille. Before joining Johns Hopkins University, he obtained his Bachelor of Science degree in Computer Science from Shanghai Jiao Tong University in 2016. His research interest is computer vision with focuses on image recognition with limited data, object detection, and instance segmentation. He also worked at YITU, Baidu IDL, Adobe, and Google.